

TEXTO DE APOIO



AULA 4

Programação de Sistemas I

Professor Regiane Moreno



Universidade Presbiteriana
Mackenzie





Sumário



MÉTODOS CONSTRUTORES	3
CONSTRUTOR PADRÃO	3
CONSTRUTOR PERSONALIZADO	5
SOBRECARGA (OVERLOAD)	9
STATIC	12
REFERÊNCIAS:	14

PROGRAMAÇÃO DE SISTEMAS I

MÉTODOS CONSTRUTORES

CONSTRUTOR PADRÃO

Os construtores são métodos responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida.

Quando usamos a palavra reservada **new**, estamos construindo um objeto. Sempre que **new** é chamado, o método construtor da classe também é automaticamente chamado. O método construtor de uma classe deve ser definido com o mesmo nome da classe e, caso não exista, o Java usará o construtor **default** (sem argumentos/parâmetros).

O construtor só pode ser chamado no momento da criação de um objeto, por meio do operador **new**. O objetivo principal de um construtor é inicializar objetos novos quando são criados e executar outros procedimentos de configuração.

Portanto, sempre que um objeto é criado, automaticamente, o método construtor é executado. O construtor só pode ser chamado no momento da criação de um objeto.

O objetivo de um método construtor é inicializar objetos novos quando são criados e executar outros procedimentos iniciais de configuração.

```
Produto prod1 = new Produto();
```

└─> **Método construtor**

Nesse exemplo, o construtor vazio (sem argumentos) está sendo ativado, assim, o compilador executará o construtor vazio (numéricos inicializados com zero, boolean com false e char/String com null).

Se nenhum construtor é definido para uma classe, o compilador cria um construtor default que não recebe nenhum argumento (também chamado de construtor sem argumentos).

Observe que o operador new cria uma instância de um determinado objeto:

```
Produto prod1 = newProduto();
```



Foi criada uma referência para o objeto Televisao que se chama teste. Com ela, podemos executar os métodos da classe Televisao e modificar seus atributos.

A definição de um construtor se parece com a definição de um método, porém, existem três exceções:

- **O método construtor não tem nenhum tipo de retorno (não é permitido o comando return).**
- **O nome do método construtor tem de ser igual ao nome da classe em que está sendo definido.**
- **Os modificadores de acesso só podem ser: public, private ou protected.**

Quando nenhum método construtor é formalmente declarado, o método construtor default (padrão) é usado.

Lembra deste exemplo?

```
public class Aluno {  
  
    int codigo;  
  
    String nome;  
  
    int idade;  
  
    float nota1, nota2;  
  
    public double calcMedia() {  
  
        return (nota1 + nota2)/2;  
    }  
}
```

```
}
```

```
}
```

Nele, ao instanciar um objeto do tipo Aluno, usamos o construtor default, pois, na classe Aluno, não havia um construtor escrito pelo programador.

```
aluno1 = new Aluno();
```

CONSTRUTOR PERSONALIZADO

Se o programador quiser instanciar um objeto com valores específicos, ele terá de descrever o construtor. É necessário formalizar a declaração do método construtor com a assinatura e implementação que desejamos.

Etapas de um método construtor:

- Aloca espaço suficiente na memória para armazenar o objeto.
- Inicializa os atributos do objeto. Caso não tenha sido atribuído um valor inicial, os valores default serão usados.
- Os possíveis parâmetros informados na chamada do construtor são processados e atribuídos aos parâmetros recebidos pelo construtor.
- Os comandos do corpo do construtor são executados.
- Uma referência ao objeto é retornada como um valor da chamada do construtor.

Vejamos a classe Aluno, com um construtor personalizado.

```
public class Aluno {
```

```
    String nomeAluno;
```

```
    double n1,n2,n3;
```

```
    Aluno(String nome) {
```

```
        nomeAluno = nome;
```

Método construtor



```
}
```

← Método construtor

```
}
```

Na classe que tem o método `main()`, para instanciar um objeto usando este construtor, devemos escrever:

```
Aluno estudante = new Aluno("José da Silva");
```

Veja outro exemplo:

```
public class ParDeDados {
```

```
    int dado1;
```

```
    int dado2;
```

```
    ParDeDados(int val1, int val2) {
```

```
        dado1 = val1;
```

```
        dado2 = val2;
```

```
    }
```

← Método construtor

```
    public void jogarDados () {
```

```
        dado1 = (int)(Math.random() * 6) + 1;
```

```
        dado2 = (int)(Math.random() * 6) + 1;
```

```
    }
```

O exemplo acima mostra a declaração do método construtor para o objeto ParDeDados. Sabe-se que esse é o método construtor porque o nome é exatamente igual ao nome da classe. No exemplo, quando um objeto do tipo ParDeDados for criado, seus atributos serão automaticamente inicializados com os valores passados como parâmetros (valor1 e valor2). No método main(), a criação do objeto ParDeDados poderia estar assim:

```
ParDeDados jogada = newParDeDados(1,1);
```

Nesse caso, quando o objeto ParDeDados for criado, seus atributos dado1 e dado2 serão automaticamente inicializados com o valor 1 (que foi passado como parâmetro).

CLÁUSULA THIS

É comum que o método construtor, escrito pelo programador (personalizado), use nos parâmetros desse método a mesma grafia dos atributos e, para resolver essa ambiguidade de nomes, é necessário o uso do this.

Veja um exemplo de método construtor escrito pelo programador (personalizado), no qual os parâmetros do construtor têm mesma grafia que os atributos.

```
public class Aluno {
```

```
    String nome;
```

```
    double n1,n2,n3;
```

```
    Aluno(String nome) {  
        this.nome = nome;  
    }
```

← **Ambiguidade de nomes**

```
}
```

A variável `this` é uma referência para o próprio objeto e deve ser usada nas seguintes situações:

- Em um construtor, pode-se executar outro construtor que tenha uma assinatura diferente. Para isso, a execução do comando de chamada para outro construtor deve ser o primeiro comando executado no método.
- Resolver ambiguidade de nome entre um atributo e um parâmetro ou da variável local de algum método.
- Retornar a própria referência da instância em alguns métodos.

Exemplo:

```
public class Conta {  
  
    int agencia;  
  
    int conta;  
  
    Conta(){  
  
        this(0,0);  
  
    }  
  
    Conta(int agencia, int conta){  
  
        this.agencia = agencia;  
  
        this.conta = conta;  
  
    }  
}
```



```
public alteraConta (int conta) {  
  
    this.conta = conta;  
  
    return this;  
  
}  
  
...  
  
}
```

SOBRECARGA (OVERLOAD)

É a capacidade de definir mais de um método com o mesmo nome, porém com assinaturas diferentes em uma única classe. Isso pode acontecer também com o método construtor. Para sobrecarregar um método de uma classe, simplesmente forneça uma definição separada de método com o mesmo nome para cada versão do método. Lembre-se de que os métodos sobrecarregados devem ter listas de parâmetro diferentes.

Caso, na classe em que tiver um construtor personalizado, o programador queira também o construtor padrão, ele deverá escrevê-lo.

O compilador chama o construtor apropriado comparando a quantidade, os tipos e a ordem dos argumentos especificados na chamada do construtor com a quantidade, os tipos e a ordem dos parâmetros especificados em cada definição de método.

Implementaremos o projeto com a classe Aluno, escrevendo alguns construtores nessa classe:

```

1  public class Aluno {
2      int codigo;
3      String nome;
4      int idade;
5      float notal, nota2;
6
7      Aluno() {
8      }
9
10
11     Aluno(int codAluno) {
12         codigo = codAluno;
13     }
14
15
16     Aluno(int codigo, String nome) {
17         this.codigo = codigo;
18         this.nome = nome;
19     }
20
21     public double calcMedia() {
22         return (notal + nota2)/2;
23     }
24
25     public void mostraDados() {
26         System.out.println("Codigo = " + codigo + "\nNome = " + nome + "\nIdade = " + idade);
27         System.out.println("Nota 1 = " + notal + "\nNota 2 = " + nota2);
28         System.out.println("");
29     }
30 }

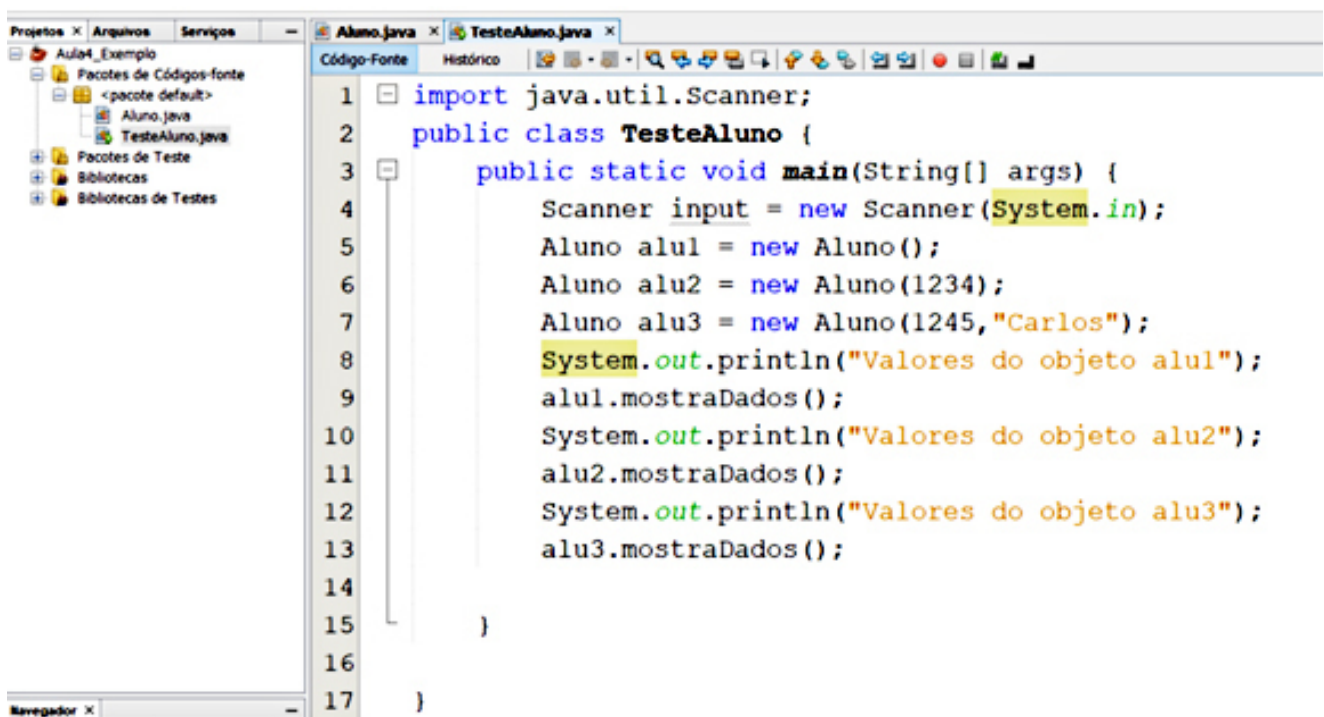
```

Construtor padrão

Construtor personalizado

Construtor com ambiguidade de nomes

Agora, instanciaremos objetos diferentes, usando diferentes construtores:



```

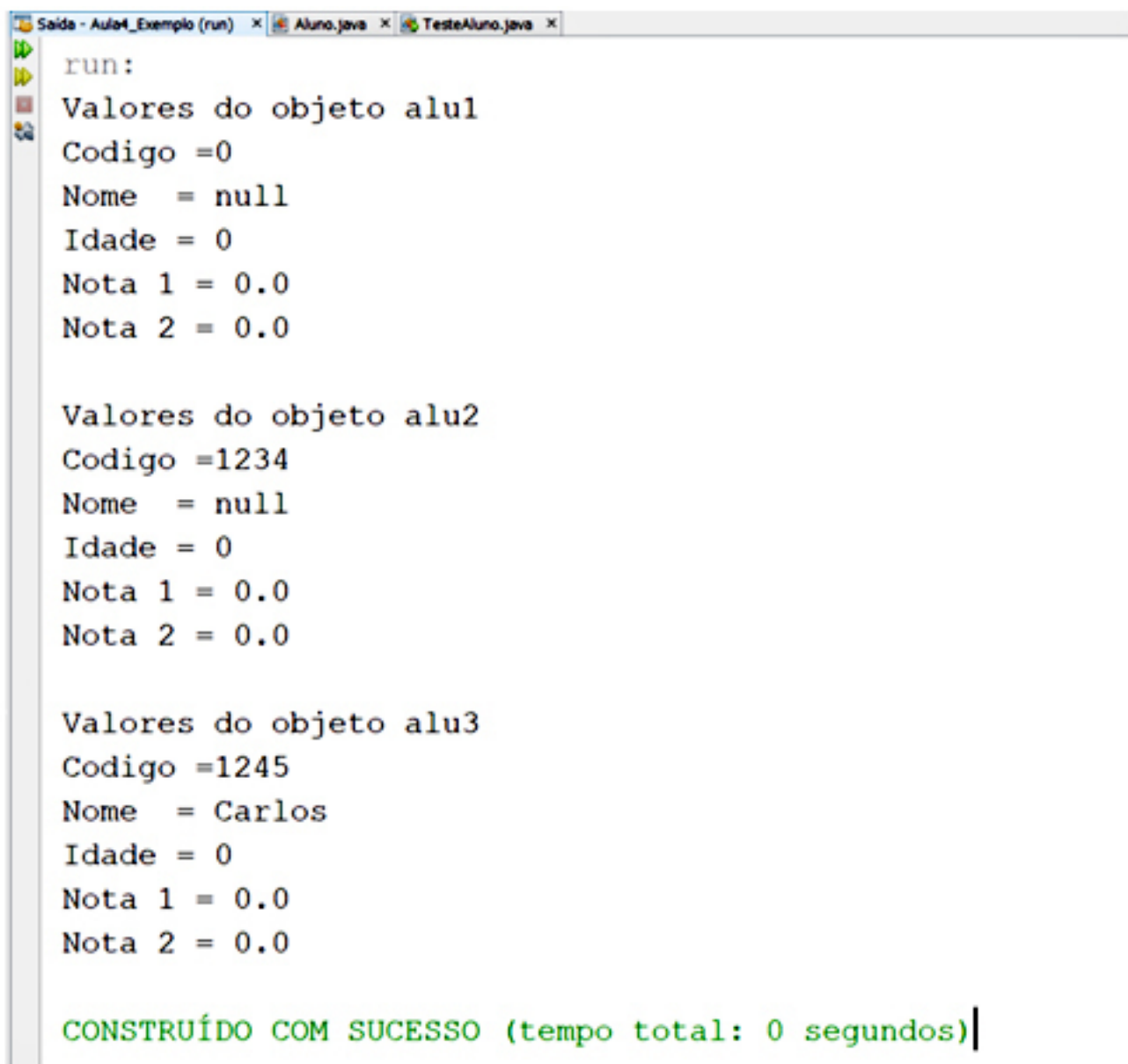
1  import java.util.Scanner;
2  public class TesteAluno {
3      public static void main(String[] args) {
4          Scanner input = new Scanner(System.in);
5          Aluno alu1 = new Aluno();
6          Aluno alu2 = new Aluno(1234);
7          Aluno alu3 = new Aluno(1245, "Carlos");
8          System.out.println("Valores do objeto alu1");
9          alu1.mostraDados();
10         System.out.println("Valores do objeto alu2");
11         alu2.mostraDados();
12         System.out.println("Valores do objeto alu3");
13         alu3.mostraDados();
14
15     }
16
17 }

```

Observe, no código acima, que o objeto alu1 foi instanciado por meio do construtor default (padrão), o objeto alu2 foi instanciado com o construtor personalizado que recebe como parâmetro o código do aluno, e o objeto alu3 foi instanciado com o construtor que recebe como parâmetro o código e nome do aluno.

Os atributos que não foram inicializados de forma personalizada pelo construtor terão seu valor inicial default.

Observe:



```
run:
Valores do objeto alu1
Codigo =0
Nome  = null
Idade = 0
Nota 1 = 0.0
Nota 2 = 0.0

Valores do objeto alu2
Codigo =1234
Nome  = null
Idade = 0
Nota 1 = 0.0
Nota 2 = 0.0

Valores do objeto alu3
Codigo =1245
Nome  = Carlos
Idade = 0
Nota 1 = 0.0
Nota 2 = 0.0

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)|
```

construtor da classe.

STATIC

Até agora, você percebeu que usamos o objeto para acessar os atributos e métodos do modelo. Por exemplo, acima, para executar o método `mostraDados()`, usamos cada um dos objetos instanciados.

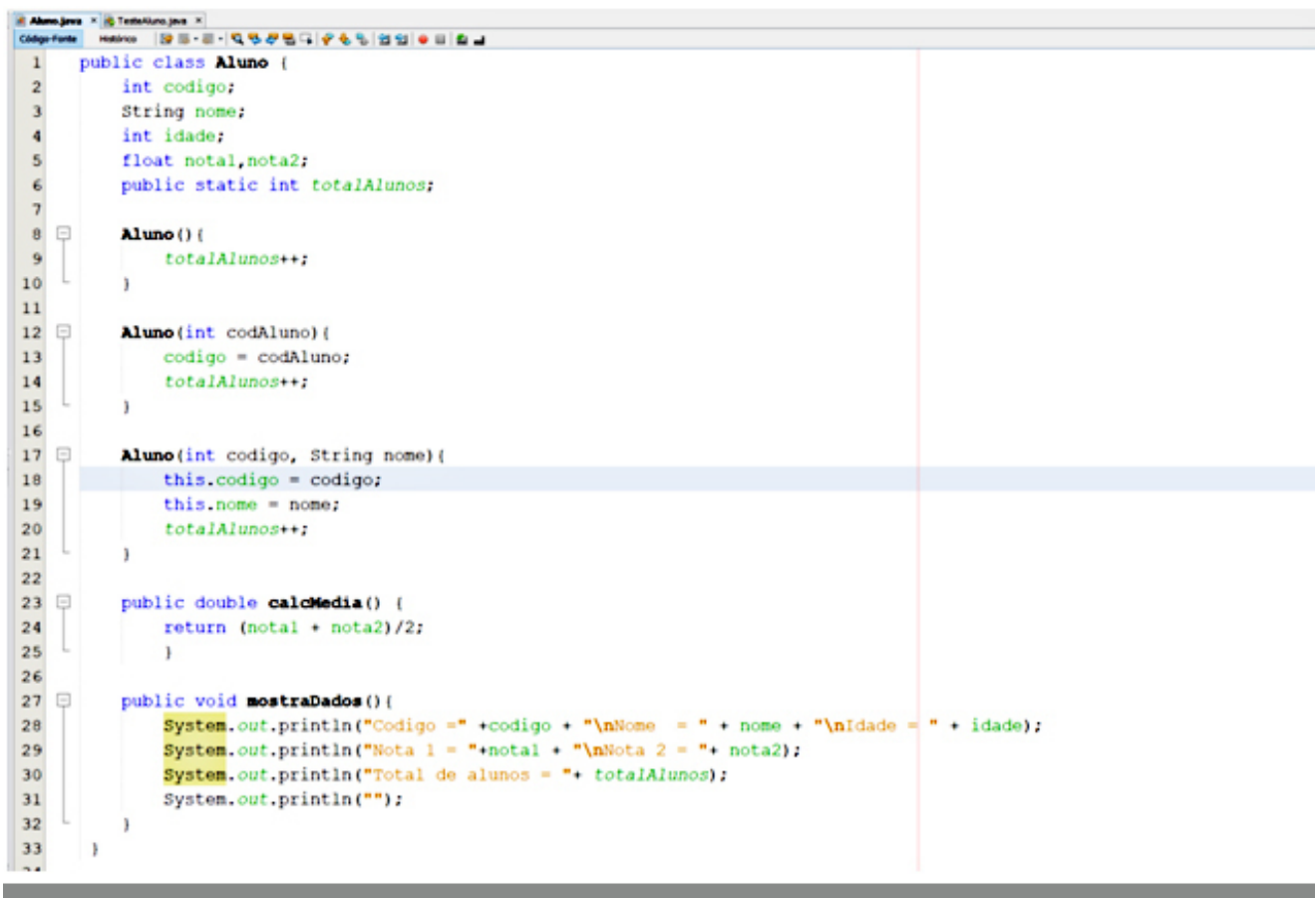
Quando definimos atributos com a palavra `static` em uma classe, ele terá um comportamento especial: ele será o mesmo para todos os objetos daquela classe.

Ou seja, não haverá um tipo dele em cada objeto. Todos os objetos, ao acessarem e modificarem esse atributo, acessarão a mesma variável, o mesmo espaço da memória, e a mudança poderá ser vista em todos os objetos.

Sintaxe:

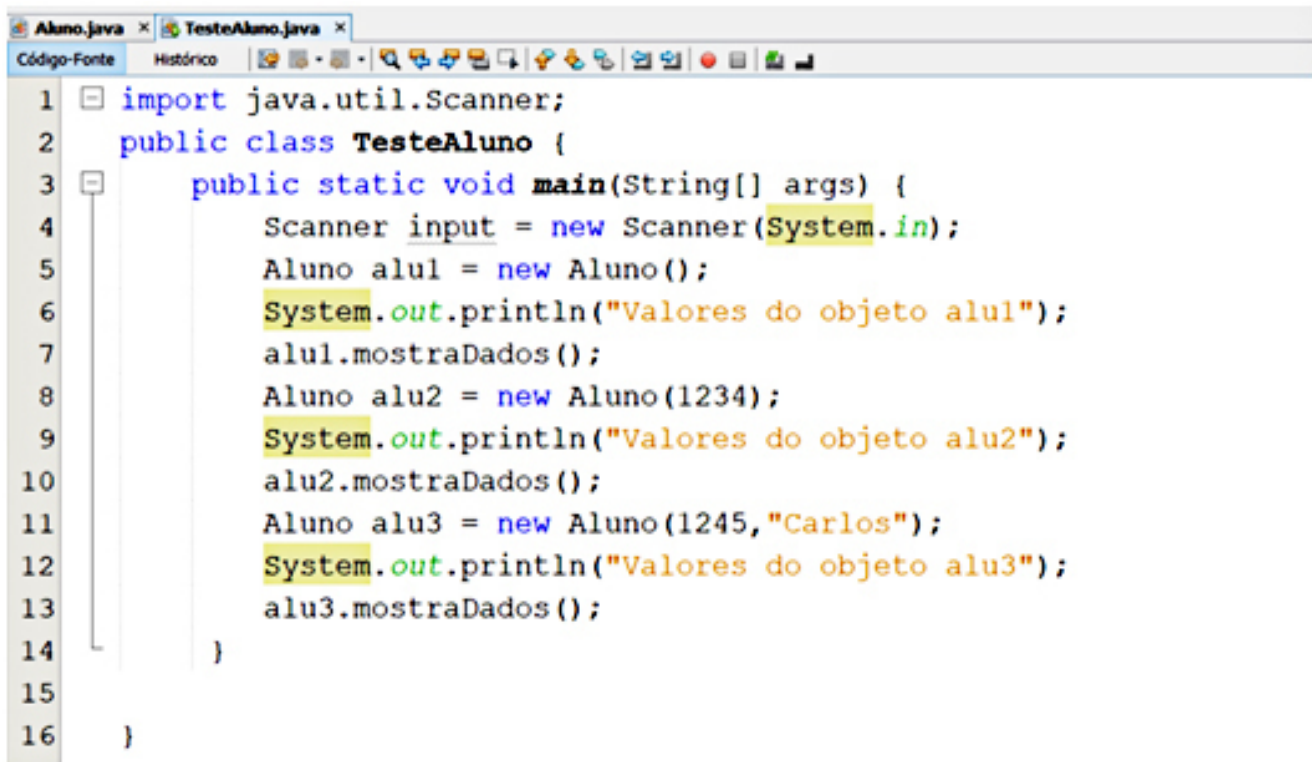
`static<<tipo>>nomedoAtributo;`

Veja o exemplo com um atributo estático na classe `Aluno`, usado para contar as instâncias:



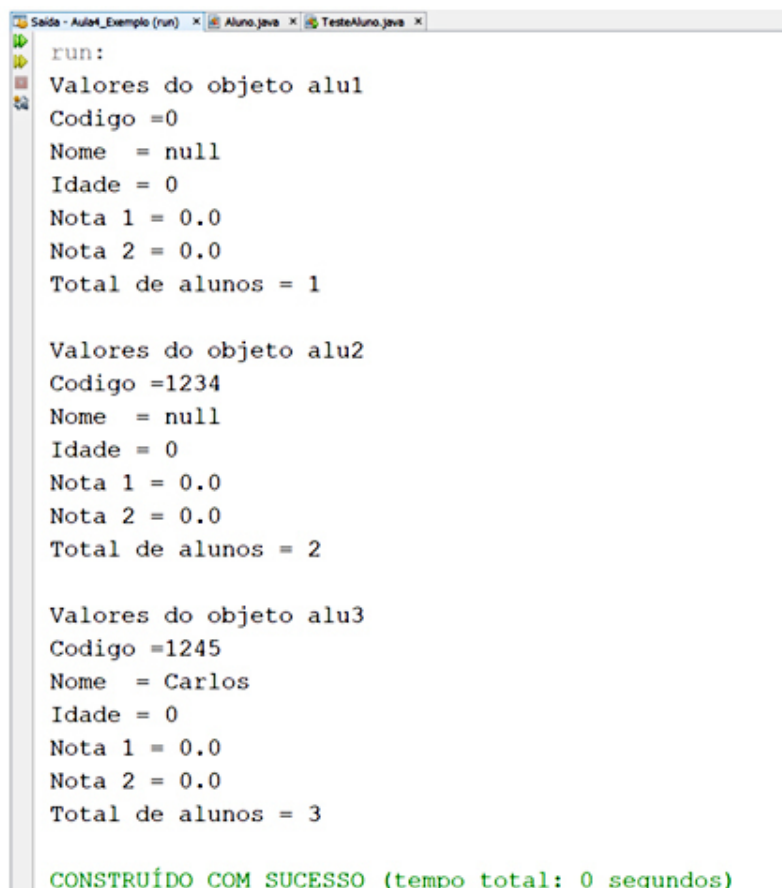
```
1 public class Aluno {
2     int codigo;
3     String nome;
4     int idade;
5     float nota1, nota2;
6     public static int totalAlunos;
7
8     Aluno() {
9         totalAlunos++;
10    }
11
12    Aluno(int codAluno) {
13        codigo = codAluno;
14        totalAlunos++;
15    }
16
17    Aluno(int codigo, String nome) {
18        this.codigo = codigo;
19        this.nome = nome;
20        totalAlunos++;
21    }
22
23    public double calcMedia() {
24        return (nota1 + nota2) / 2;
25    }
26
27    public void mostraDados() {
28        System.out.println("Codigo = " + codigo + "\nNome = " + nome + "\nIdade = " + idade);
29        System.out.println("Nota 1 = " + nota1 + "\nNota 2 = " + nota2);
30        System.out.println("Total de alunos = " + totalAlunos);
31        System.out.println("");
32    }
33 }
```

Na classe com o método main(), foram instanciados três objetos e, a cada objeto instanciado, foi invocado o método mostraDados():



```
1 import java.util.Scanner;
2 public class TesteAluno {
3     public static void main(String[] args) {
4         Scanner input = new Scanner(System.in);
5         Aluno alu1 = new Aluno();
6         System.out.println("Valores do objeto alu1");
7         alu1.mostraDados();
8         Aluno alu2 = new Aluno(1234);
9         System.out.println("Valores do objeto alu2");
10        alu2.mostraDados();
11        Aluno alu3 = new Aluno(1245, "Carlos");
12        System.out.println("Valores do objeto alu3");
13        alu3.mostraDados();
14    }
15
16 }
```

Veja, a cada chamada do método mostraDados(), o atributo estático totalAlunos fazendo a função de contador.



```
run:
Valores do objeto alu1
Codigo =0
Nome = null
Idade = 0
Nota 1 = 0.0
Nota 2 = 0.0
Total de alunos = 1

Valores do objeto alu2
Codigo =1234
Nome = null
Idade = 0
Nota 1 = 0.0
Nota 2 = 0.0
Total de alunos = 2

Valores do objeto alu3
Codigo =1245
Nome = Carlos
Idade = 0
Nota 1 = 0.0
Nota 2 = 0.0
Total de alunos = 3

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

REFERÊNCIAS:

DEITEL, H.; DEITEL, P. Java – Como Programar. 10ª. ed. São Paulo: Pearson - Prentice-Hall, 2017.

HORSTMANN, C. S.; CORNELL, G. Core Java. 8ª. ed. São Paulo: Pearson Prentice Hall, 2010.