3 – Classes e instanciação de objectos (em Java)

Suponhamos que queremos criar uma classe que especifique a estrutura e o comportamento de objectos do tipo Contador.

As instâncias da classe Contador devem verificar o seguinte:

1 – os contadores são do tipo inteiro;

2 – ser possível criar contadores com

2.1 – valor inicial igual a zero;

2.2 – valor inicial igual a um dado valor dado como parâmetro;

3 – ser possível incrementar o contador

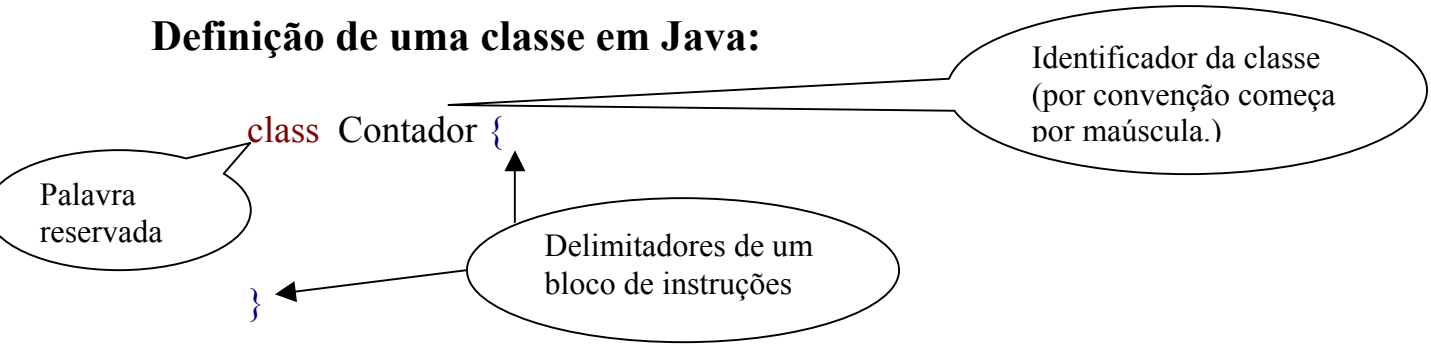
3.1 – de uma unidade;

3.2 – de dado valor dado como parâmetro;

4 – o mesmo para decrementar;

5 – ser possível obter uma representação textual do contador.

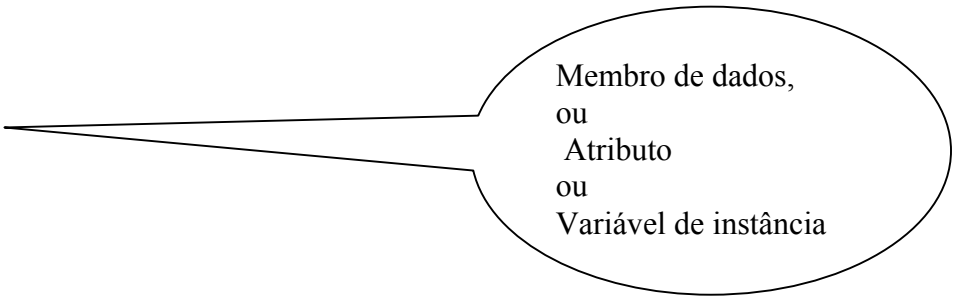
Definição de uma classe em Java:



Definindo a classe Contador:

1º estrutura interna – (uma variável do tipo int)

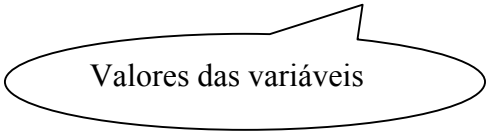
```
class Contador {
        int conta;
        ...
}
```



2º definição do comportamento (operações ou métodos)

2.1 – Construtores

Um construtor é um método "especial" que permite inicializar o estado das instâncias da classe




```
/*... já voltamos à classe Contador  ...*/
```

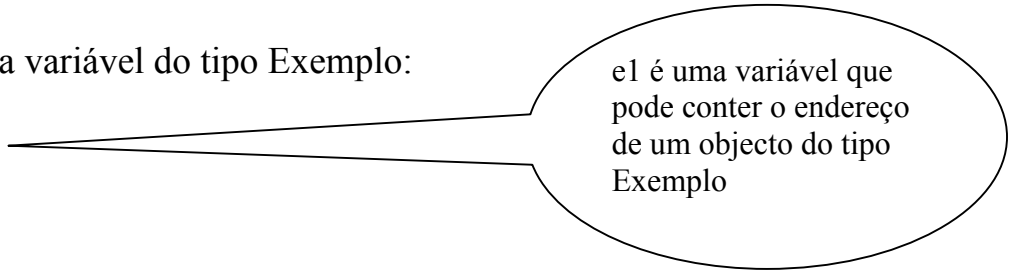
Suponhamos a classe Exemplo:

```
class Exemplo {
        int i; // um atributo que é um tipo primitivo
        ClasseA a; // um atributo que é um objecto do tipo ClasseA
        ...
}
```

Para criar uma instância da classe exemplo (isto é criar um objecto do tipo Exemplo) temos de:

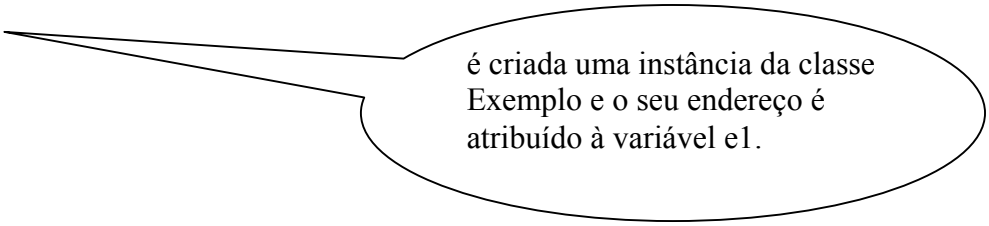
1) Declarar uma variável do tipo Exemplo:

```
Exemplo e1;
```

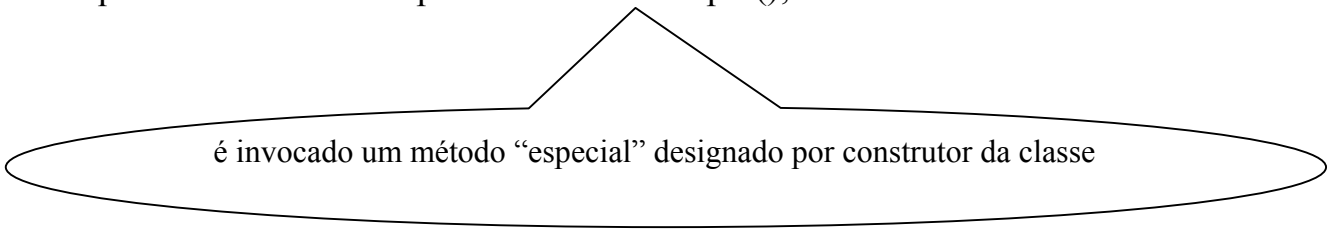


2) Instanciar o objecto do tipo exemplo referenciado por e1:

```
e1 = new Exemplo();
```



1) e 2) são equivalentes a `Exemplo e1 = new Exemplo();`



Por omissão (isto é, se não for definido nenhum outro) toda a classe possui um construtor.

Este construtor atribui o valor nulo (null) a todas as variáveis do objecto que sejam do tipo referenciado (objectos e arrays*). As variáveis dos tipos primitivos são inicializadas com os seguintes valores:

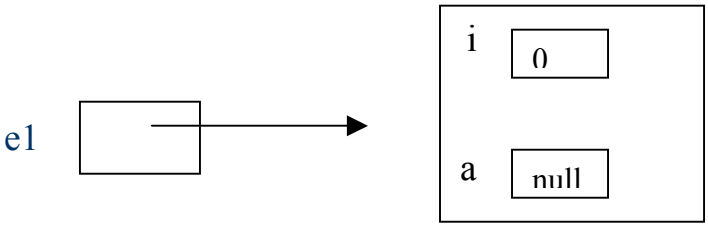
(* os arrays depois de instanciados são inicializados de acordo com o seu tipo)

inteiros (byte, short, int e long) – 0

reais (float e double) – 0.0

carácter (char) – (char)0

lógico (boolean) – false

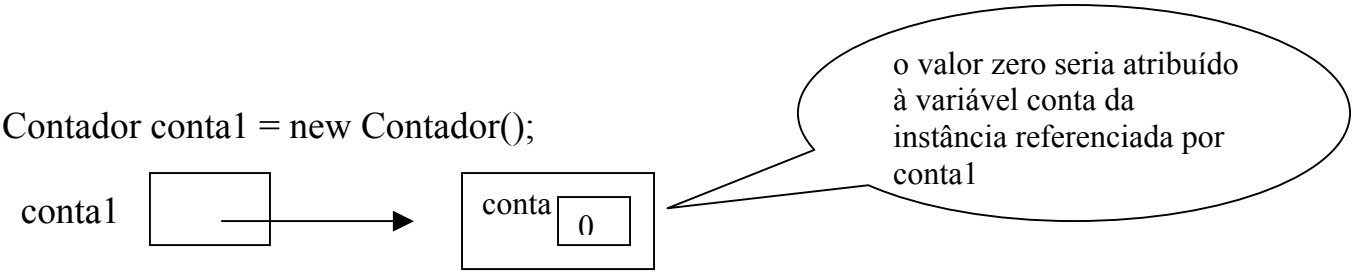


O método construtor por omissão pode ser redefinido com um outro comportamento:

```
/*... regressando à classe Contador  ...*/

class Contador {

  int conta;

 Contador () {

     conta = 0;   // atribuição explícita do valor inicial da variável conta

 }

...

}
```

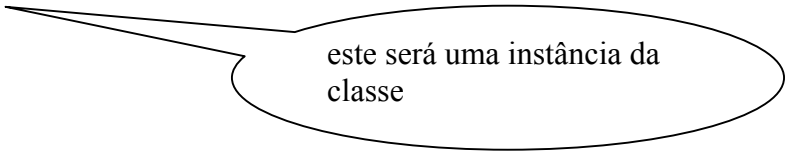
Se criarmos um objecto do tipo Contador:

Contador conta1 = new Contador();



São construtores de uma classe:

- todos os métodos que tenham por identificador exactamente o mesmo nome da classe,
- com qualquer nº e tipo de argumentos e
- sem especificação de resultado.

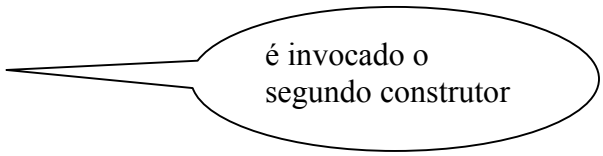


Definição de um segundo construtor para a nossa classe contador:

```
Contador (int val) {
   conta = val;                //2º construtor
}
```

Poderíamos agora criar um objecto Contador com valor inicial por exemplo igual a 100:

```
Contador conta2 = new Contador(100);
```



Os vários construtores de uma classe são diferenciados pela sua lista de parâmetros.

2.1 – Métodos de instância

Regra 1: Devemos garantir que nenhum objecto faz acesso directo às variáveis de instância de outro objecto.

(Forma de garantir que estamos a definir objectos independentes do contexto e consequentemente reutilizáveis)

Ex.lo

```
Contador contax = new Contador();
contax.conta++;  //  má programação
```

A interface do objecto deverá fornecer todos os métodos necessários para acesso ao estado do objecto.

Regra 2: Um objecto genérico não deve ter instruções de input/output no seu código.

Definição de um método:

cabeçalho ("header") ➔

<tipo do resultado> <identificador> (pares tipo e nome do parâmetro)

. qualquer tipo primitivo
. nome de um classe
. void

corpo ("body") ➔ {conjunto de instruções}

Assinatura de um método:

É constituída pelo identificador do método e pelo número, tipo e ordem dos seus parâmetros.

/*... continuando com a classe Contador ...*/

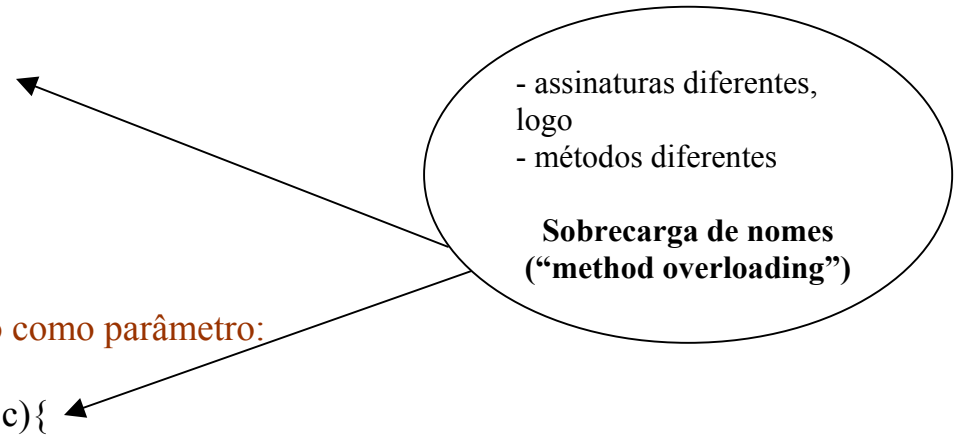
Métodos para incrementar o contador

. de uma unidade:

```
void incConta(){
    conta++;
}
```

. de um valor dado como parâmetro:

```
void incConta(int c){
    conta += c;
}
```



Para invocar os métodos:

```
...
conta1.incConta();
...
conta1.incConta(10);
...
```

Métodos para decrementar o contador

```
void decConta(){
  conta--;
}

void decConta (int c){
  conta -= c;
}
```

Método para representar o objecto sob a forma de texto:

```
   String toString() {

        return ("Contador = " +  conta );  // operador para concatenação de Strings: +

  }
```

Usar o método toString:

```
...

 String s;

Contador conta3 = new Contador (123);

s = conta3.toString();

System.out.println (s);

...
```

Classe String – classe pré-definida na linguagem

Um valor do tipo String á uma sequência de zero ou mais caracteres entre aspas.

. Um objecto do tipo String tem um valor imutável

. O operador "concatenação de Strings", +, cria implicitamente uma nova instância da classe String.

s = s + "XPTO" //de cada operação de concatenação resultará uma nova String,

//criada temporariamente, que no final será atribuída à variável s.

São equivalentes as instruções:

String nome = new String ("XPTO");

String nome ="XPTO";

Também no método toString seriam equivalentes:

return ("Contador = " + conta);

return (new String ("Contador = " + conta);

———

Convenção:

Métodos que façam acesso de leitura ao valor de uma variável x, designam-se por:

getX - interrogadores ou selectores ("getters")

- devolvem um resultado do tipo da variável x.

Métodos que alterem o valor de uma variável x, designam-se por:

setX - modificadores ("setters")

- geralmente têm parâmetros de entrada e não devolvem qualquer resultado.

Exemplo:

Definir na classe Contador um método para consultar o valor da variável conta:

```
 int getConta(){
   return conta;
}
```

Definir na classe Contador um método para dar um novo valor à variável conta:

```
void setConta( int c){
   conta = c;
}
```

Usar os métodos:

```
 int c = conta1.getConta();
…
conta1.setConta(c+2);
…
```

A referência this:

A abordagem da comunicação por mensagens pode ser usada uniformemente,

- quer para interacção com outros objectos
- quer para invocação de métodos locais.

Para que um objecto possa enviar uma mensagem a si próprio, terá que existir uma forma de auto-referência:

this - identificador especial que contém o endereço do próprio objecto em cujo contexto é usado.

Há situações em que é útil esta forma de auto-referência.

Ex.lo:

```
class Exemplo {
    int x,y;
   Exemplo ( int x, int y ) {
      this.x = x;
      this.y = y;
  }
  void auxiliar(){
     …
  }
  void metodo2 (){
    this. x = …;
    this. auxiliar();
  }
```

Declaração local ao método, que se sobrepõe à global.

Por omissão (em Java)

Packages em Java

- As classes são agrupadas de acordo com a sua funcionalidade.
- Cada classe de um package tem acesso às outras classes do mesmo package.

Exemplos:

package java.io

- conjunto de classes que implementam funcionalidades relacionadas com input/output

package java.util

- funcionalidades de uso geral

inclui as classes: Date, GregorianCalendar, EventListener, Vector, ...

package java.lang

- classes fundamentais à execução de programas

inclui as classes: String, Boolean, Character, Float, Integer, System, …

O nome absoluto de uma classe ou método tem como prefixo o nome do package:

java.lang.String.length();

java.lang.System.out.println();

…

Se queremos aceder a classes de outros packages:

- colocamos o nome completo (absoluto)

ou

- usamos uma cláusula de importação

```
import java.util.Vector;
import java.util.*;
```

Todas as classes do package

import java.lang.*; → implicitamente inserida em todos os packages definidos pelo utilizador

Usar uma cláusula de importação ("import") não significa que as classes do package vão ser copiadas para o package do nosso programa. Serve apenas para podermos referir o nome de uma classe ou método na sua forma abreviada omitindo o nome do package a que pertence.

(Para usarmos classes de outros packages definidos por nós será necessário ou incluir esses packages numa biblioteca ou definir a variável de ambiente CLASSPATH com a directoria onde estão as classes).