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 valor dado como parâmetro;

3 – ser possível incrementar o contador

3.1 – de uma unidade;

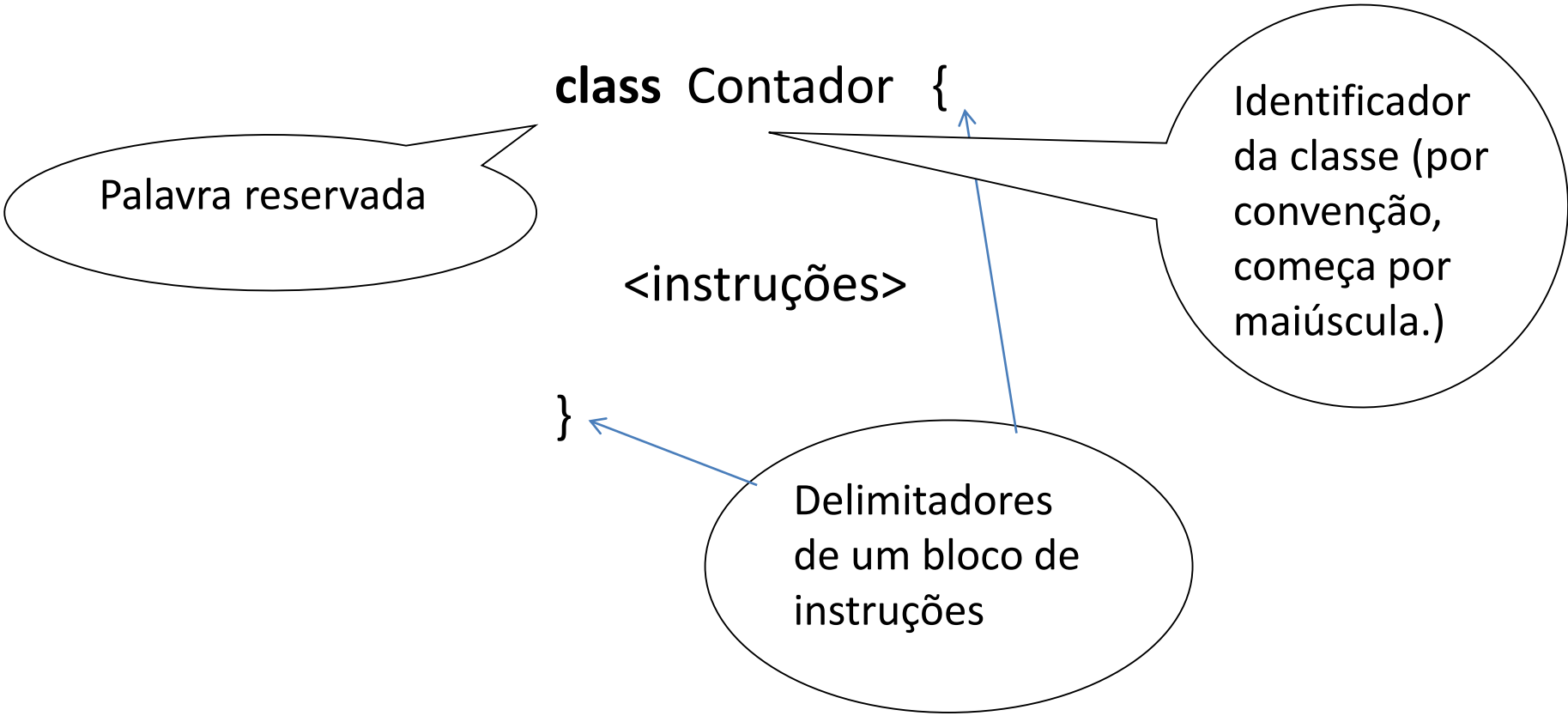
3.2 – de um dado valor dado como parâmetro;

4 – o mesmo para decrementar;

5 – ser possível obter uma representação textual do contador.

??
O que é um parâmetro?
Qual a diferença entre argumento e parâmetro?

Definição de uma classe em Java:



Definindo a classe Contador:

1º, definimos a estrutura do novo tipo de dados

Neste caso, a classe tem apenas um atributo, uma variável do tipo int.

```
class Contador {

        int conta;

        ...
}
```

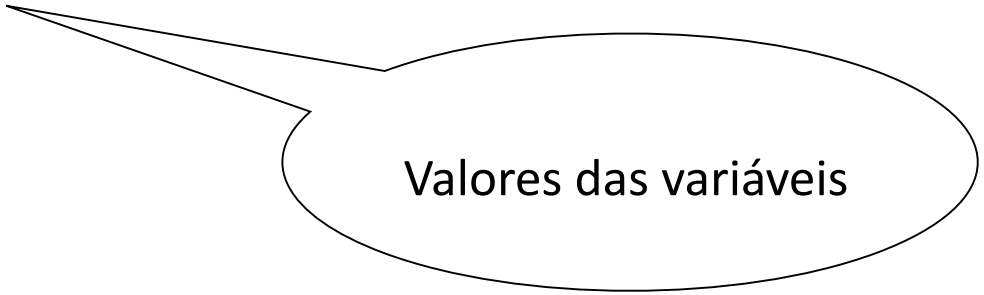
?? Qual a diferença entre uma classe e uma instância?

Definindo a classe Contador:

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
        ...
}
,
```

?? Qual a diferença entre declarar uma variável e inicializar uma variável?

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

- Declarar uma variável do tipo Exemplo:

1) Exemplo e1;

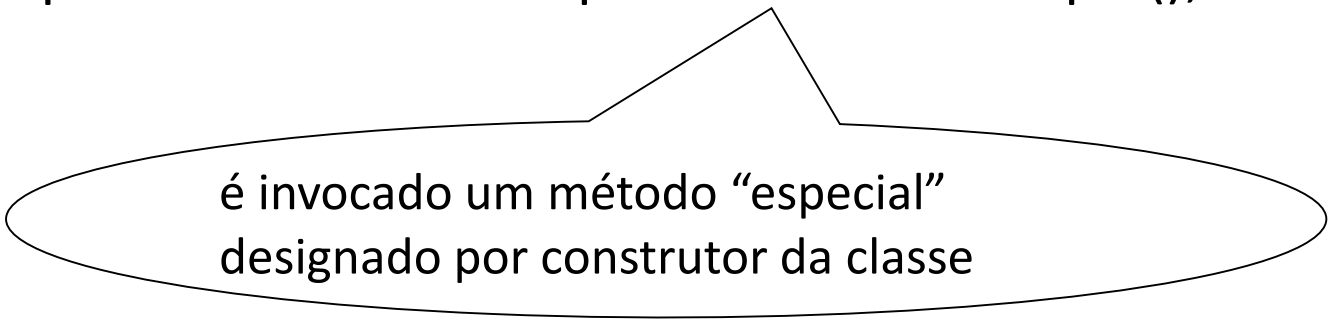
e1 é uma variável que pode conter o endereço de um objecto do tipo Exemplo

-Instanciar o objecto do tipo Exemplo referenciado por e1:

2) e1 = new Exemplo();

é criada uma instância da classe Exemplo e o seu endereço é atribuído à variável e1.

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.

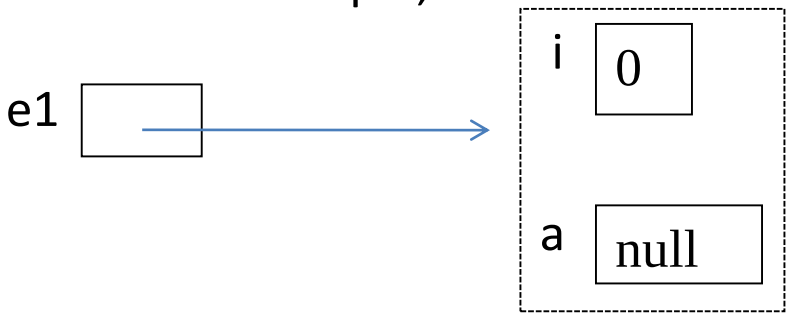
- Este construtor atribui o valor nulo (null) a todas as variáveis do objecto que sejam do tipo referenciado (objectos e arrays*).

- As variáveis dos tipos primitivos são inicializadas com os seguintes valores:

- inteiros (byte, short, int e long): 0
- reais (float e double): 0.0
- carácter (char) (char)0
- lógico (boolean) false

os arrays, depois de instanciados, são inicializados de acordo com o seu tipo

- No caso da classe Exemplo, seria.



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

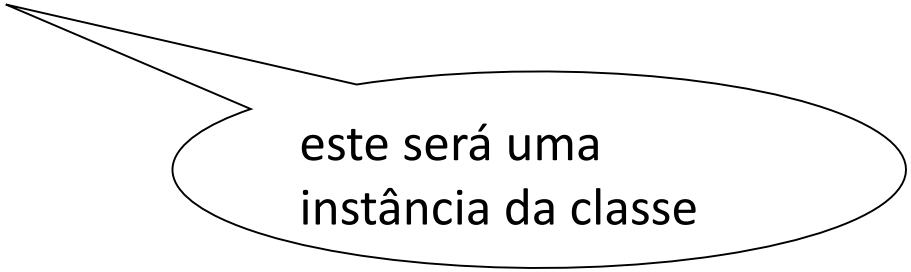
```
/*... regressando à classe Contador  ...*/

class Contador {

  int conta;

 Contador () {

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

São construtores de uma classe:

- todos os métodos que tenham por identificador exactamente o mesmo nome da classe,

- com qualquer número e tipo de parâmetros e

- sem especificação de resultado.



?? Qual a diferença entre declarar um objeto e instanciar um objeto?

Definição de um segundo construtor para a nossa classe contador:

```
Contador ( int val )  {

        conta = val;                          //2º construtor
}
```

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

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

é invocado o segundo construtor

- 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
. array
nome de uma classe
. void

corpo (“body”) ➔ {conjunto de instruções}

Definição

Assinatura de um método:

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

Exemplos:

int **metodo1(int x, double y)**

Assinatura: metodo1_int_double

double **metodo2(double d1, double d2)**

Assinatura: metodo2_double_double

/*... 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;
}
```

- assinaturas diferentes, logo
- métodos diferentes

Sobrecarga de nomes

("method overloading")

Para invocar os métodos :

… main (…)

```
Contador conta1, conta2;  //declaração de dois objetos do tipo Contador

conta1 = new Contador();  //instanciar o objeto conta1

conta2 = new Contador (100)

conta1.incConta();  //invocar um método no objeto conta1

conta1.incConta(10);

conta2.incConta ();
```

?? Quais as variáveis que existem neste programa??

??Qual o seu valor no final?

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);
...
System.out.println(  conta3.toString() )
```

?? Qual o output deste bloco de código?

Classe String – classe pré-definida na linguagem

Um valor do tipo String é 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”); //invocar o construtor d classe String

String nome =”XPTO”; // atribuir o valor diretamente

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")
- têm um parâmetro de entrada do tipo da variável x, e não devolvem qualquer resultado.

Exemplo:

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

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

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

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

Usar os métodos:

```
Contador conta1 = new Contador();

 int c = conta1.getConta();
…

conta1.setConta(c+2);
…

String s = conta1.toString();

System.out.println (s);
```

Output= ??

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.

Exemplo:

```
class Exemplo {
    int x, y;
    Exemplo  ( int x, int y) {
        this.x = x;
        this.y = Y;
    }
```

Declaração local ao método, que se sobrepõe à global.

```
void auxiliar(){
        …
}
void metodo2 (){
        this.  x = …;

        this.  auxiliar ();
}
```

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:

ArrayList, Vector, LocalDate, EventListener, ...

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.ArrayList;
```

```
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).

Mecanismos de controlo de acesso

Especificam "quem" tem acesso a cada entidade, isto é, quem tem acesso a cada classe e cada membro da classe (dados e métodos)

Modificadores de acesso:

public

protected

private

"por omissão"

Regras de acesso a classes:

R1: Uma classe é sempre acessível a todas as outras classes do mesmo package
(qualquer que seja o modificador de acesso).

R2: Se nenhum modificador de acesso é usado, a classe apenas pode ser acedida dentro do seu package.

R3: Quando uma classe é declarada como "public" pode ser acedida por qualquer classe que tenha acesso ao seu package.

R4: Quando uma classe é não pública apenas é acessível dentro do seu package.

Regras de acesso a variáveis e métodos:

"norma:"

variáveis são privadas,
métodos de interface são públicos,
métodos auxiliares são privados

R1: Um método declarado como "public" é acessível de qualquer ponto de qualquer programa.

Designa-se por API ("Application Programming Interface") de uma classe, o conjunto de métodos de instância que não forem declarados como "private".

R2: Um método sem modificador de acesso é acessível a qualquer classe do mesmo package.

R3: Métodos ou variáveis declarados como “private” são apenas acessíveis dentro da própria classe.

R4: Métodos ou variáveis declarados como “protected” são acessíveis na própria classe, de outra classe dentro do mesmo package e nas subclasses da classe (!).

Definição da classe Contador

```
public class Contador  {

        // variáveis de instância
        private int conta;

        // construtores
        public Contador () {
            conta = 0;
        }
        public Contador ( int conta) {
                    this.conta = conta;
        }
```

```
// métodos de instância
        public int getConta(){    return conta;  }

        public void setConta( int conta ) { this.conta= conta; }

        public void incConta () {
                conta ++;
        }
        public void incConta (int inc) {
                conta = conta + inc;
        }
        public void decConta () {
                conta --;
        }
        public void decConta (int dec) {
                conta = conta - dec;
        }
        public String toString () {
                return (“Contador: “ + conta );
        } }
```

Classes de teste

- Uma classe de teste serve para verificar toda a funcionalidade de uma classe

```
public class TesteContador {

   public static void main ( String[] args) {
```

A execução de qualquer programa em Java inicia-se por um método com este cabeçalho.

```
// criar instâncias da classe
        Contador c1, c2;
        c1 = new Contador();
        c2= new Contador (10);

// enviar mensagens às instâncias criadas, para obter o valor dos contadores

        int i1, i2;
        i1 = c1.getConta();
        i2 = c2.getConta();

// verificar os resultados
        System.out.println ("c1 = " + i1 );
        System.out.println ("c2 = " + i2 );
```

```
// alterar valores
            c1.incConta();
            c2.incConta(10);
            System.out.println ("c1= " + c1.getConta() + "\n" + "c2 = " + c2.getConta() );
            c1.decConta();
            c2.decConta (2);

// converter para String
        String s = c1.toString();
         System.out.println (s);
         System.out.println ( c2.toString() );

        System.out.println ( c2 );  //(o que acontece??)

   } // main
} // class TesteContador
```

Exercício: Qual o output deste programa?

Verifique a sua solução implementando o programa.

c1 = 0
c2 = 10
c1= 1
c2 = 20
Contador: 0
Contador: 18
Contador: 18

(??) Numa instrução de escrita, o método toString definido pelo utilizador será usado para converter o objecto para texto mesmo quando não é explicitamente invocado.

Exercícios:

1 – Construa uma classe que represente os Empregados de uma empresa. Um Empregado tem um número de segurança social, um nome e um salário. Defina os atributos, dois construtores à sua escolha, os métodos de consulta (getters) e os de modificação (setters), e o método toString. Construa ainda um método que permita subir o salário do empregado de uma dada percentagem dada como parâmetro.

2 – Construa uma classe de teste que permita verificar todos os métodos da classe Empregado