

Variáveis e Métodos de Classe

Em Java, quer as classes quer as instâncias das classes são objectos.

Onde está o estado da classe?

Com que operações é manipulado?

Variáveis de classe

- representam a estrutura interna de uma dada classe

Métodos de classe

- métodos que implementam o comportamento da classe.

São invocados através de mensagens enviadas à classe.

Como se declaram?

- com o identificador `static`

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Ex.lo

```
public static int metodoA()
```

Método de
classe

```
private static String texto;
```

Variável de
classe

Para que servem?

Usam-se variáveis de classe para armazenar valores que digam
“respeito” a todos os objectos da classe.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Exemplo:

Se quiséssemos saber, em determinado instante, quantos objetos do tipo Contador já tinham sido instanciados.

```
public class Contador {  
    // declarar uma variável de classe que vai conter o n° de  
        objetos instanciados  
  
    private static int contadores = 0;  
  
    // método de classe  
  
    public static int getContadores () {  
        return (contadores) ;  
    }  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
// variável de instância  
private int conta;
```

```
// reescrever os construtores
```

```
public Contador () {  
    conta = 0;  
    contadores++;  
}
```

```
public Contador ( int conta) {  
    this.conta = conta;  
    contadores++;  
}
```

...

Sempre que é criada uma instância da classe Contador a variável **contadores** é incrementada.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

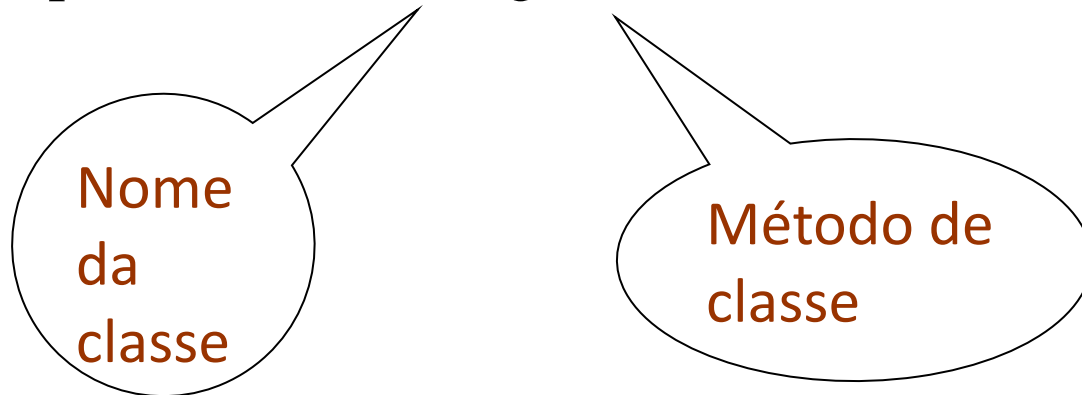
Na classe TesteContador:

```
public class TesteContador {  
  
    public static void main ( String[] args) {  
  
        System.out.println (“Nº de objetos do tipo Contador” );  
  
        System.out.println (Contador.getContadores());  
  
        Contador c1 = new Contador();  
        Contador c2 = new Contador(10);  
  
        // Que variáveis existem neste ponto do programa e quais os  
        // seus valores?
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
System.out.println (“Nº de objetos do tipo Contador” );
```

```
System.out.println (Contador.getContadores());
```



```
}
```

O output do programa anterior é?

Variáveis de classe

- podem ser usadas mesmo que nunca tenha sido instanciado um objeto da classe.

Métodos de classe

- são acessíveis às instâncias da classe, isto é, um método de instância pode invocar um método de classe.

Métodos de classe não podem invocar métodos de instância.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Situações anteriores em que já usou valores ou métodos de classe?

...

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

```
System.out ;
```

```
System.in ;
```

```
Calendar.HOUR_OF_DAY;
```

```
Math.random();
```

...

Classes não instanciáveis:

- São classes só com variáveis e métodos de classe

(diferente de classes com apenas uma instância)

- Não especificam a estrutura nem o comportamento de qualquer instância.
- Representam “centros de serviços” que não faz sentido replicar

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Exemplo:

Classe Math:

Math.mensagem();

double x, y, z;

...

double x = Math.sqrt(y);

double y = Math.cos (x);

double z = Math.random();

...

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Exercício:

Suponha que um aluno, tem um número, um nome e um conjunto de disciplinas a que está inscrito.

Construa a classe Aluno, considerando que sempre que um novo aluno é criado, o número de aluno será atribuído automaticamente de forma sequencial.

Composição de Classes

Quando definimos uma classe, alguns atributos podem ser objectos de qualquer outra classe já definida.

Suponhamos a classe Telefone:

```
public class Telefone {  
    private String tipo; // Casa | Emprego | Móvel | ...  
    private int numero;  
  
    public Telefone () {  
        this.tipo = "";  
        this.numero = 0;  
    }  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public Telefone (String tipo, int numero){
    this.tipo = tipo;
    this.numero = numero;
}
public String getTipo (){
    return tipo;
}
public int getNumero (){
    return numero;
}
public void setTipo (String tipo){
    this.tipo = tipo;
    //tipo é uma variável local ao método
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public void setNumero (int numero){  
    this.numero = numero;  
    // numero é uma variável local ao método  
}  
  
public String toString (){  
    String s = "Tipo: " + tipo + " Número: " + numero;  
    return s;  
}  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Definimos agora a classe Pessoa em que cada Pessoa tem um nome e um Telefone:

(Ex.io: Generalizar para cada Pessoa ter vários telefones)

```
public class Pessoa {  
    private Telefone tel;  
    private String nome;
```


Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
// construtor
```

```
public Pessoa (String nome) {  
    this.nome = nome;  
    tel = new Telefone();    ← instanciar o objecto telefone  
}
```

```
public void setTelefone(Telefone t){  
    tel = t; // O que acontece ???  
}
```

```
// t é local ao método, mas t é uma variável que apenas  
// contém o endereço de um objecto do tipo telefone
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public String toString (){  
    return "Nome : " + nome + "\nTelefone - "  
        + tel ;    // o que acontece ???  
}  
}
```

Suponhamos a classe Teste: ➔

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Teste {  
    public static void main (String[] args ) {  
  
        Telefone t1 = new Telefone ("Casa", 275123456);  
  
        Pessoa p = new Pessoa ( "Marco António");  
  
        p.setTelefone(t1);  
  
        System.out.println(p.toString()); //(1)  
  
        // Qual o output?
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Nome : Marco António

Telefone - Tipo: Casa Número: 275123456

// Se modificarmos a variável t

```
t1.setTipo("Emprego");
```

```
t1.setNumero(111111111);
```

// O que acontece na Pessoa p ???

```
System.out.println(p.toString()); //(2)
```

```
}
```

```
}
```

Qual é o output?

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Nome : Marco António (1)

Telefone - Tipo: Casa Número: 275123456

Nome : Marco António (2)

Telefone - Tipo: Emprego Número: 111111111

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Se na classe Pessoa, modificarmos o método setTelefone para:

```
public void setTelefone(Telefone t){  
    tel.setTipo( t.getTipo());  
    tel.setNumero( t.getNumero());  
    // O que acontece ???  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

O output do Programa anterior passará a ser:

Nome : Marco António (1)

Telefone - Tipo: Casa Número: 275123456

Nome : Marco António (2)

Telefone - Tipo: Casa Número: 275123456

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Notas:

Em Java os parâmetros são passados por valor.

É criada uma variável local com valor igual a uma cópia do argumento.

Se o parâmetro é um tipo referenciado (objecto ou array), equivale à passagem por referência.

O argumento é a referência de um objecto ou array.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Exercício: considere a classe Valor

```
public class Valor {  
    private int val;  
  
    public int getVal () {  
        return val;  
    }  
  
    public void setVal (int v) {  
        val = v;  
    }  
}
```

Qual o output do programa →

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Teste2 {  
  
    public static void main(String[] args) {  
        int i1 = 3;  
        int i2 = i1;  
        i2 = 4;  
  
        System.out.println( "i1 = " + i1 );  
  
        System.out.println(" mas i2 = " + i2 );  
    }  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

...

```
Valor v1 = new Valor();
```

```
Valor v2 = new Valor();
```

```
v1.setVal(5);
```

```
v2 = v1; // o que acontece???
```

```
v2.setVal(6);
```

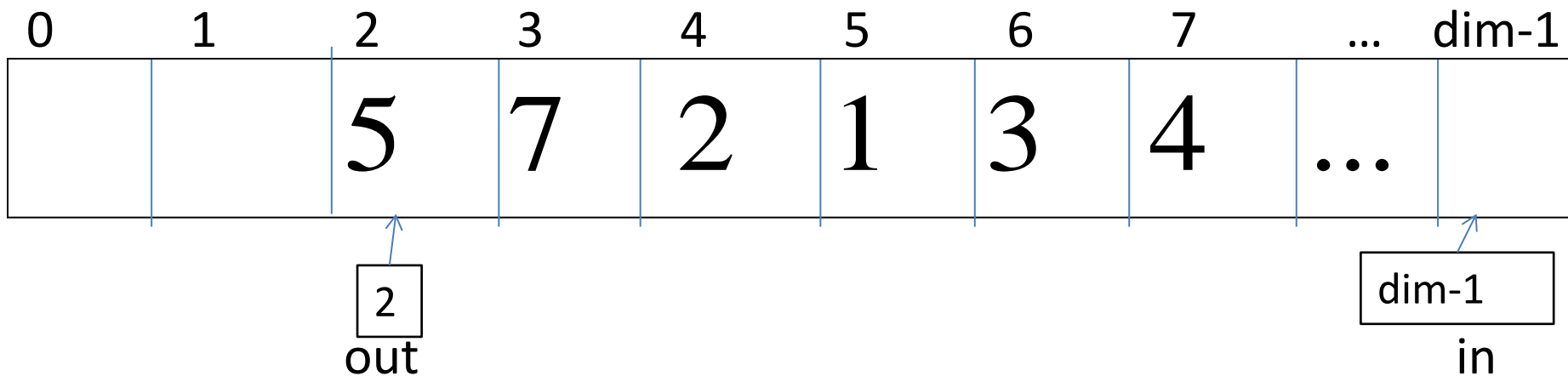
```
System.out.println( " v1.val " + v1.getVal() );
```

```
System.out.println( " v2.val " + v2.getVal() );
```

```
}  
}
```

Métodos parciais e pré-condições

Suponhamos uma classe que implementa o tipo abstracto de dados Fila (First In First Out) usando um array circular de valores inteiros.



out – posição do próximo elemento a sair

in – posição para onde entra o próximo elemento

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

out – posição do próximo elemento a sair

in – posição para onde entra o próximo elemento

Fila cheia: $in = out$

Fila vazia: $in == out$

Para distinguir: *variável com o número de elementos da fila.*

n_elementos

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public class Fila {  
  
    // variáveis de instância  
  
    private int [] fila ;  
    private int n_elementos, dim, in, out;  
  
    // construtor  
    public Fila (int n){  
        dim = n;  
        fila = new int[dim];  
        in = 0;  
        out = 0;  
    }  
}
```

// métodos de instância

```
public void inserir ( int e ){  
    fila [in] = e;  
    n_elementos ++;  
    in = (in + 1) % dim;  
}
```

Operações condicionadas

- nem sempre podem ser realizadas

```
public int retirar (){  
    int e = fila [out];  
    n_elementos --;  
    out = (out + 1) % dim;  
    return e;  
}  
}
```

Como resolver?

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Hipótese 1:

- colocar um parâmetro de retorno que indica o sucesso da operação

```
public boolean inserir ( int e ){
```

```
    if ( n_elementos != dim ) {
```

```
        ...
```

```
        return true;
```

```
    } else {
```

```
        return false;
```

```
    }
```

```
}
```


Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Ao invocar o método:

...main

```
Fila filaA = new Fila(10);
```

```
boolean sucesso = filaA.inserir(123);
```

```
if ( ! sucesso )
```

```
    System.out.println ( “Erro: Fila Cheia” );
```

O que fazer no caso da operação retirar? Já temos um resultado!
Podemos definir um novo tipo de objecto ...

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Hipótese 2:

- implementar métodos que funcionam como pré-condições das operações condicionadas.

definir na classe Fila:

```
public boolean cheia () {  
    return (n_elementos == dim);  
}
```

```
public boolean vazia () {  
    return ( n_elementos == 0);  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Ex.lo de utilização:

...main

```
Fila filaA = new Fila (10);
```

```
if ( ! filaA.cheia() ) {
```

```
    filaA.inserir(123);
```

```
}else {
```

```
    System.out.println ( “Erro: Fila Cheia” );
```

```
}
```

```
...
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
...  
  
if ( ! filaA.vazia() ) {  
  
    int i = filaA.retirar();  
  
}else {  
    System.out.println ( “Erro: Fila vazia” );  
}  
  
...
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Construa a classe Pessoa. Uma pessoa tem um nome, um número de identificação fiscal, um conjunto de contactos (array de objetos do tipo telefone) – 3 no máximo: fixo, móvel e emprego).

- Defina um construtor que receba o nome como parâmetro;
- Defina os getters;
- Defina os setters;
- Defina o método toString;
- Defina um método que consulte o número do telefone móvel da pessoa

- Construa uma classe de teste, que crie duas pessoas e teste todos os métodos da classe Pessoa
- Construa um método que dado um array de pessoas devolva o número de pessoas que têm telemóvel.