

Linguagem de Programação III (Java Avançado)

Revisão de Conceitos Básicos*

(*Adaptação do material do Prof. César Rocha)

<http://www.dsc.ufcg.edu.br/~nelson/unipe/lp3>

Nelson Alves da Nóbrega Júnior

nelson.junior.br@gmail.com

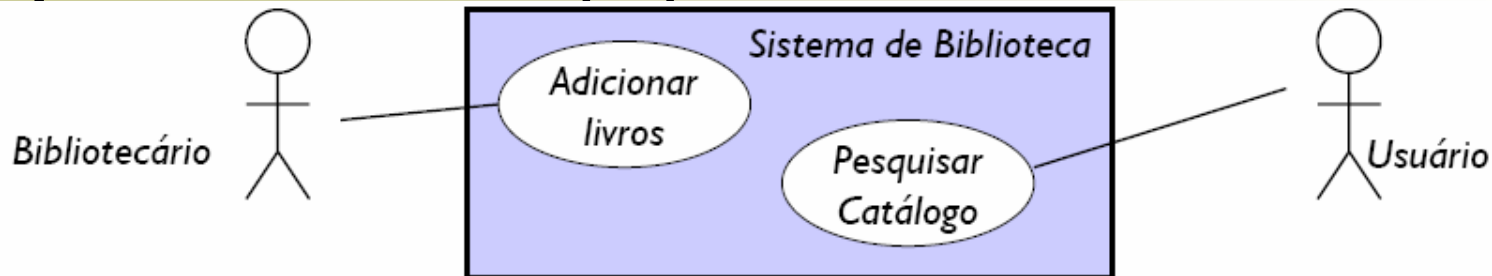
Objetivos

- Apresentar os **pré-requisitos** importantes relacionados ao curso e à Java
 - Programação orientada a objetos, estruturada, classes, objetos, estado, comportamento, encapsulamento, etc...
- Exercícios e exemplos de **códigos** que você deve testar e solidificar seus conhecimentos
 - Você pode instalar e usar qualquer IDE
- **Tentaremos ser breve**, abordando assuntos já vistos

Programação estruturada vs. POO

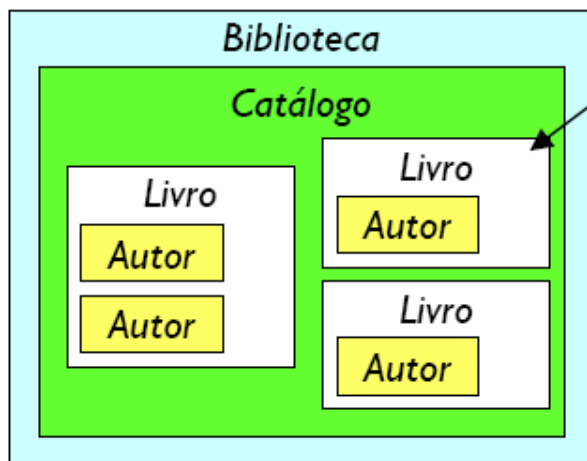
- Programação **estruturada**
 - Ênfase em **procedimentos** e **funções**
 - Modela-se a solução de um problema com base nas **funções** a serem executadas
 - Dados são tratados de forma secundária
- Programação **orientada a objetos**
 - Modelagem com base em **objetos** necessários
 - Objetos são caracterizados através de **propriedades** (informações que deve armazenar) e **comportamento** (tarefas que deverá desempenhar)

Análise OO (I) e Análise procedural (II)



(1) Trabalha no **espaço do problema** (casos de uso simplificados em objetos)

- Abstrações mais simples e mais próximas do **mundo real**

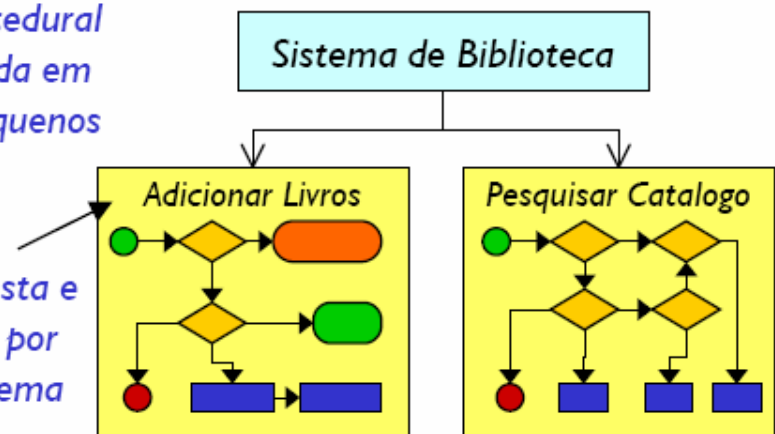


Lógica procedural encapsulada em objetos pequenos

Lógica exposta e espalhada por todo o sistema

(2) Trabalha no **espaço da solução** (casos de uso decompostos em procedimentos algorítmicos)

- Abstrações mais próximas do **mundo do computador**



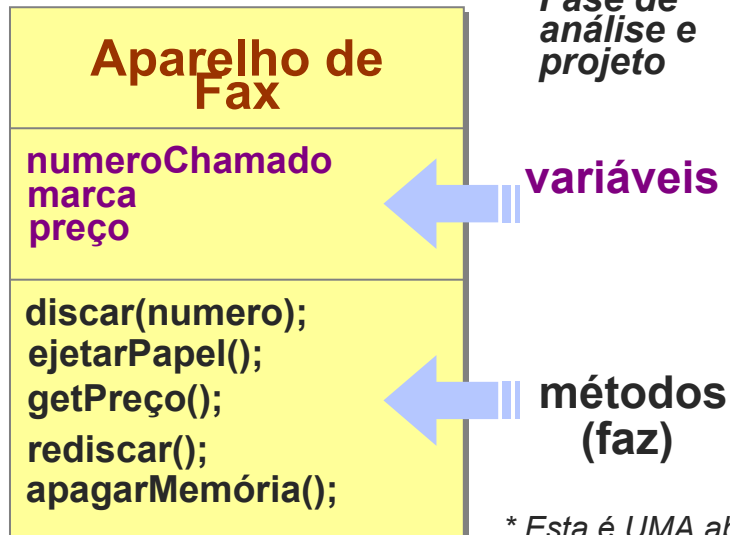
Como fazer a modelagem inicial?

■ **Classes** descrevem:

- Os dados que compõem os objetos (**variáveis**)
- Procedimentos que o objeto poderá executar (**métodos**)



UML



Fase de
análise e
projeto

Fase de
implementação

Código em
Java

```
(...)
AparelhoFax af;
af = new AparelhoFax();
af.discar("555-5555");
af.rediscar();
af.ejetarPapel();
(...)
```

Annotations in the diagram:

- An arrow points from 'Classe Java (tipo)' to the first line 'AparelhoFax af;'.
- An arrow points from 'Referência' to the variable 'af'.
- An arrow points from 'Criação do objeto' to the 'new' keyword in 'af = new AparelhoFax();'.
- An arrow points from 'Envio de mensagem' to the method call 'af.discar("555-5555");'.

* Esta é UMA abstração, em vez de A abstração

Sintaxe de classes

- Estrutura fundamental de programação em Java!
 - Todo e qualquer programa Java deve definir pelo menos uma classe. Não há como escrever código Java sem que haja a definição de classes.
 - **Sintaxe:**

*Código em
Java*

```
<modificador> class <nomeDaClasse>  
{
```

ATRIBUTO(S)

CONSTRUTOR(ES)

MÉTODO(S)

```
public class CdPlayer {  
    int faixaAtual;  
    String nomeMusica;  
  
    public CdPlayer() { }  
  
    public void tocar() { }  
    public void parar() { }
```

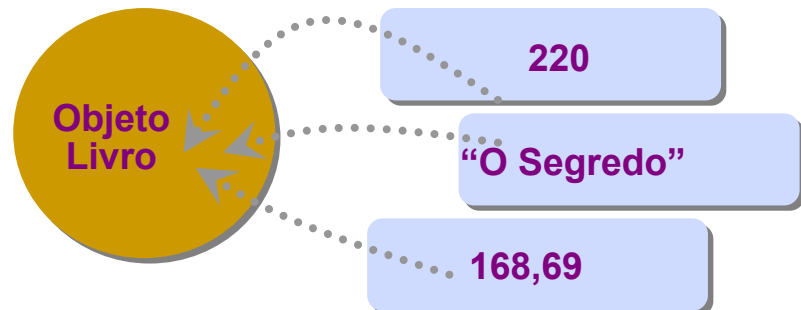
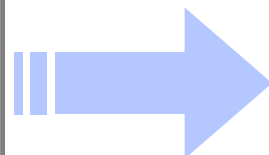
Classes e Objetos

- Classes especificam a estrutura e o comportamento dos objetos
- Classes são como “moldes” para a criação de objetos
- Objetos são instâncias de classes

/ Classe Livro */*

```
public class Livro {  
    int numPaginas;  
    String titulo;  
    double preco;  
};
```

... new Livro();



Atributos

- Definição formal:
 - “O estado de um objeto consiste de todas as **propriedades** do objeto mais os **valores** atuais destas propriedades [Booch]”
 - Cada atributo tem um **tipo** e armazena um **valor** que pode variar ao longo do tempo (é dinâmico!)

Lembre-se: *A quantidade de propriedades não muda!*
Os valores guardados é que são dinâmicos!
*Atributos devem ser **encapsulados**!*

Tipos Primitivos



boolean (8 bits)



byte (8 bits)



char (16 bits)



short (16 bits)



int (32 bits)

long (64 bits)



float (32 bits)

double (64 bits)



Tipos de Referência

- Os objetos, em um programa Java, são manipulados por variáveis chamadas de **referências**
 - Como Java é uma linguagem **fortemente tipada**, estas variáveis devem ser declaradas e tipificadas em tempo de compilação (uma classe ou interface)
 - Pode-se pensar numa variável referência como um **controle remoto*** para o objeto que pode acessá-lo e ativar seus serviços

Tipos de Referência

■ Ilustrando

```
public class UsaLivro {  
    public static void main(String[] args) {  
        Livro liv;  
  
        liv = new Livro();  
        liv.setTitulo("O Sol");  
        liv.setPreco(69.90);  
  
        liv = null; //coletor lixo  
    }  
}
```



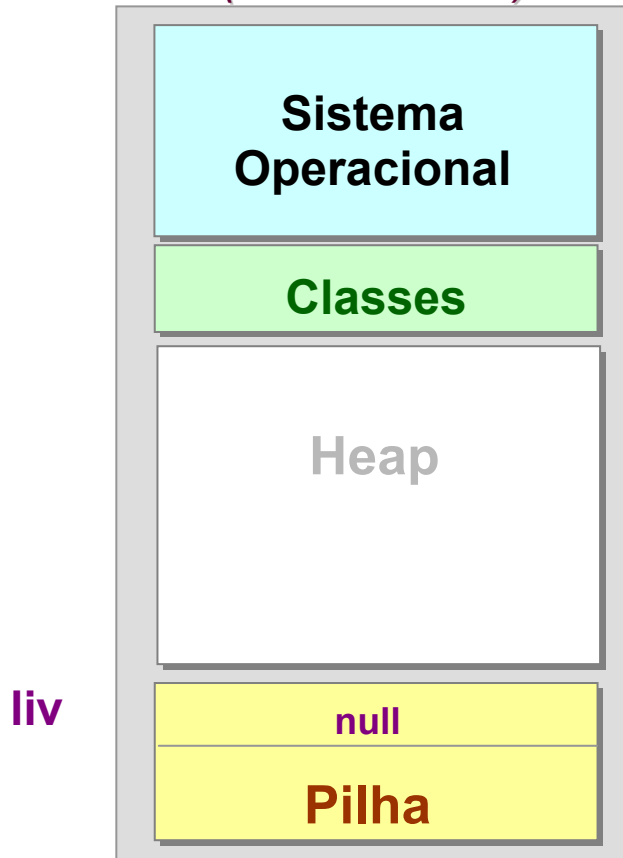
null



Tipos de Referência: Detalhes

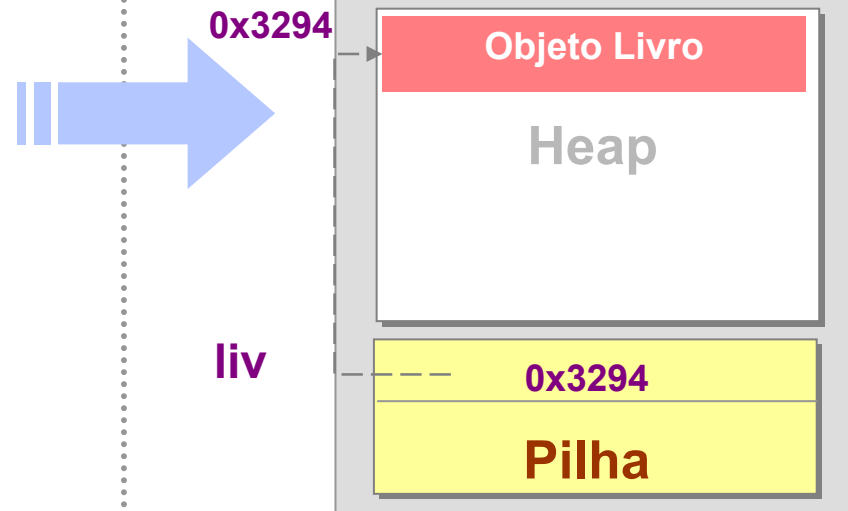
Livro liv;

Abre-se espaço na pilha para a referência (variável local)



liv = new Livro();

Reserva espaço de memória no heap de memória e atribui endereço à referência



Tipos primitivos vs. referência

■ Variáveis de tipos primitivos

Pilha após linha 2

letraPri	'a'
letraPri2	'a'

Pilha após linha 3

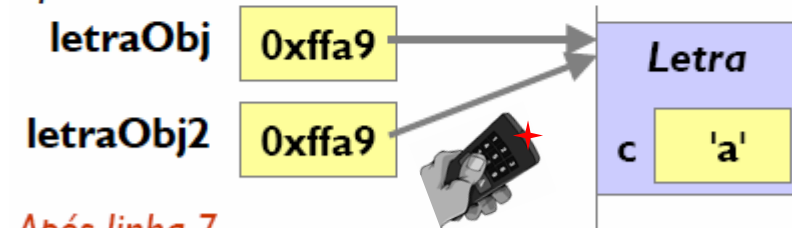
letraPri	'b'
letraPri2	'a'

(...)

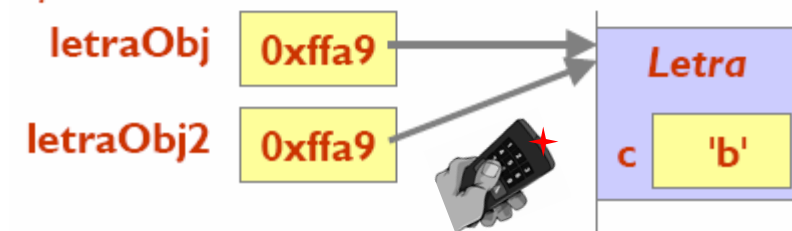
```
1: char letraPri = 'a';  
2: char letraPri2 = letraPri;  
3: letraPri = 'b';  
(...)
```

■ Referências de objetos

Após linha 6



Após linha 7



```
public class Letra {  
    public char c;  
}
```

(...)

```
4: Letra letraObj = new Letra();  
5: letraObj.c = 'a';  
6: Letra letraObj2 = letraObj;  
7: letraObj2.c = 'b';  
(...)
```

Métodos

- **Procedimento ou função** escrito na classe que permite aos objetos desta classe executarem serviços
 - É como o objeto implementa suas funcionalidades
 - O método é uma operação que age e modifica os valores dos atributos **do objeto onde ele executa!**

```
public class Cliente {  
    int idade;  
    String nome;  
    double salario;  
  
    public void alterarIdade(int idade) {  
        this.idade = idade;  
    }  
}
```

Por que não tem o parâmetro
“código do cliente”?

(...)

Encapsulamento

- Objetos são formados de duas partes:
 - ① **Interface**: métodos declarados (visão externa)
 - ② **Implementação**: a funcionalidade interna (oculta) do objeto
- Geralmente, é interessante **proibir acesso** aos atributos de dados (e até alguns métodos)
 - Modificadores de níveis de acesso, tais como: *public*, *friendly*, *protected* e *private*
 - Impacto **será menor** para quem está usando a classe
- O papel do usuário de classes
 - Apenas saber quais os métodos, parâmetros e o que é retornado (a **interface** pública da classe).

Encapsulamento



voltar()

pausar()

executar()

avancar()

alterarHora()

ejetarDisco()

Métodos de acesso

- Com o encapsulamento faz-se necessário definir métodos de acesso em atributos:
 - Acessar/ler **get<XXX>()** ou alterar **set<XXX>()** os valores das propriedades de objetos

*Implementação
foi encapsulada
com o uso de
private*

*Método que
retorna um valor
(double)*

```
public class Livro {  
    private String titulo;  
    private double preco;  
  
    public double getPreco() {  
        return preco;  
    }  
    public void setPreco(double p) {  
        preco = (p >= 0) ? p : preco;  
    }  
}
```

interno

público

Cenário de utilização

■ Visibilidade do encapsulamento em Livro:

```
public class Livraria {  
    private String endereço;  
  
    public void cadastrarNovoLivro(String tit){  
        ...  
    }  
    public void aplicarDesconto(Livro l){  
        double precoAtual = l.preco;  
        // double precoAtual = l.getPreco();  
    }  
}
```

Código errado



Acesso proibido!

A screenshot of a Windows Command Prompt window titled "Prompt de comando". The window has a red background and displays the following text in white:


```
D:\Teste\src>javac Livraria.java  
Livraria.java:8: preco has private access in Livro  
    double precoAtual = l.preco;  
                           ^  
1 error
```

Criação de objetos

- Para criar objetos, algumas linguagens implementam certos “métodos especiais”, ou **construtores**
 - Em Java, os construtores têm o mesmo nome da classe
- A alocação de todo o objeto em memória é feita com o uso do operador **new**

Lembre-se: objetos precisam ser criados antes de serem usados

```
Livro l = new Livro();  
l.setPreco(126.80);  
double valor = l.getPreco();
```



Construtores

- Para a criação de objetos, Java **garante** que cada classe tenha ao menos um construtor
 - O **construtor default** recebe zero argumentos
 - Faz apenas a inicialização da superclasse
- Construtor default só existe quando **não há nenhum outro construtor definido** explicitamente no código
 - A criação de um construtor explícito substitui o construtor fornecido implicitamente pelo sistema

Graficamente

```
public class Livro {  
    private String titulo;  
    private double preco;  
  
    public Livro() {  
        super();  
    }  
  
    public double getPreco()  
    public void setPreco(double p)
```

Construtor default
faz apenas a
inicialização da
superclasse
(construtor de
Object)

Poderia ter outros
construtores*

```
public class Livro {  
    private String titulo;  
    private double preco;  
  
    // construtor explícito  
    public Livro(String t, double p) {  
        titulo = t; // setTitulo(t);  
        preco = p;  // setPreco(p);  
    }  
  
    public double getPreco() {...}  
    public void setPreco(double p) {...}  
    public String getTitulo() {...}  
    public void setTitulo(String t) {...}  
}
```

* Distinção é feita pelo número e tipo de argumentos (ou seja, pela assinatura do construtor)

Dica: Construtor já estaria
“em dia” com as regras de
validação definidas em
métodos de acesso.

Variáveis de classe

- *Variáveis de instância* residem dentro de objetos e possuem valores (geralmente) individuais
 - Cada vez que um objeto é criado, novas propriedades são alocadas para uso daquele objeto em particular
- Porém, às vezes, um sistema pode ter variáveis contendo informações úteis, como:
 - Número de objetos instanciados pela classe até certo instante
 - Valor médio, mínimo..

Variáveis de Classe

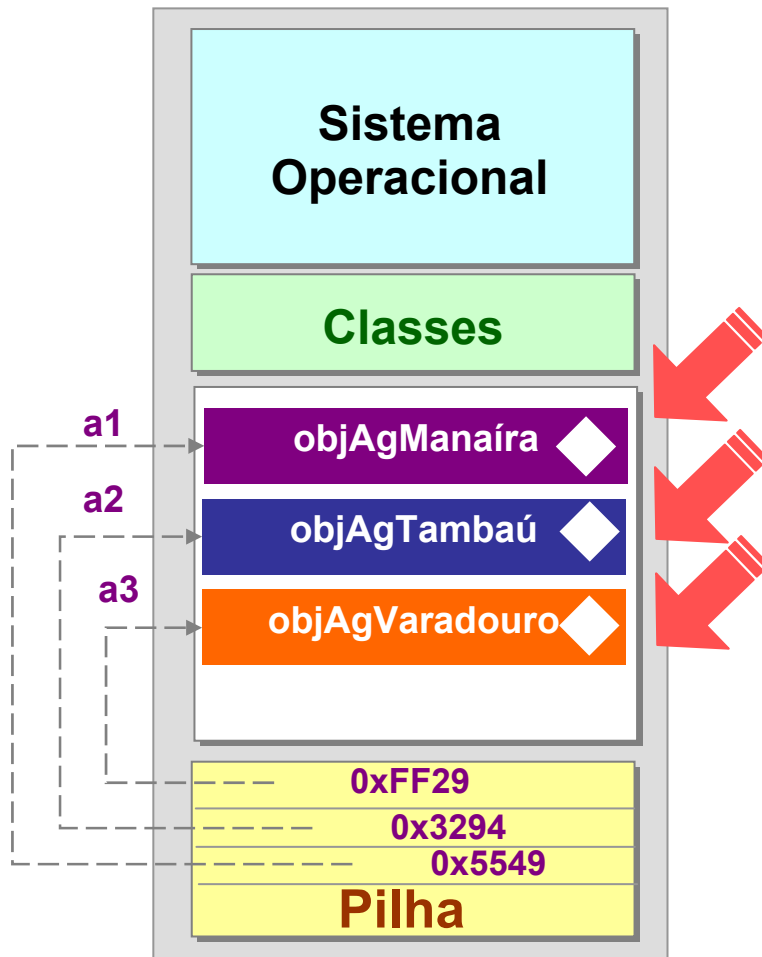
- Exemplo:
 - Taxa de juros a ser utilizada por todas as agências bancárias

```
public class Agencia {  
    private String endereco;  
    private int nome;  
    public static double juros = 0.4;  
  
    public String getEndereco() {...}  
    public void ligarAlarme() {...}  
}
```

Graficamente

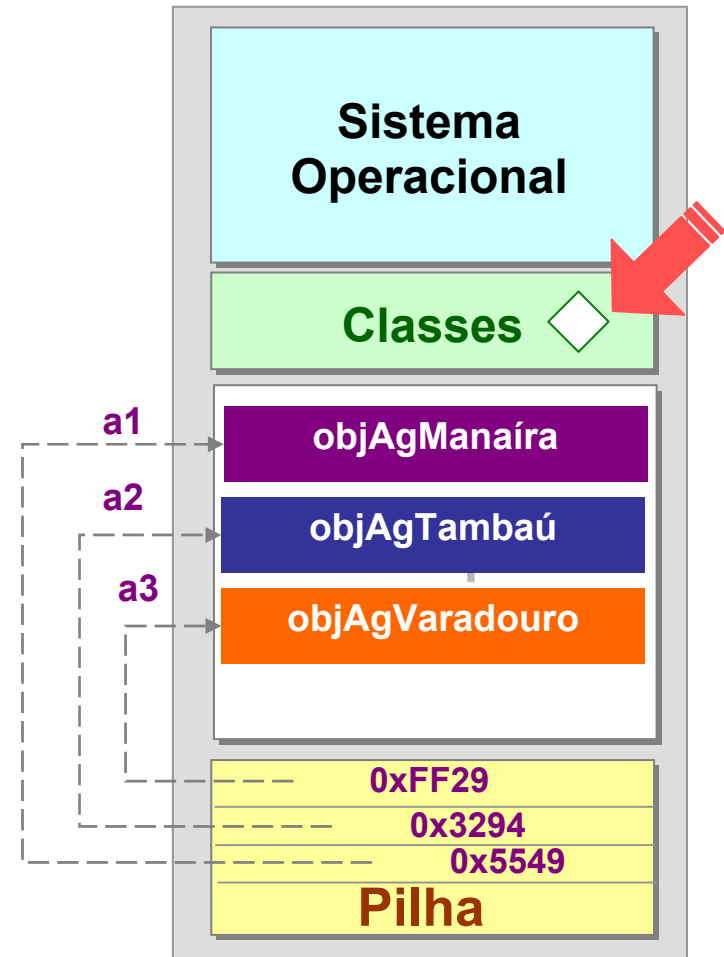
1ª tentativa

*Tornar esta variável
como propriedades das
instâncias*



2ª tentativa

*A única cópia residindo apenas na
classe (alterações apenas em um
local)*



Variáveis de Classe

- Os objetos instanciados pela classe podem acessar as variáveis de classe (para modificar ou ler o valor)
- A modificação numa variável de classe é percebida por todos os objetos
- A variável de classe fica armazenada **na classe** e não nas instâncias geradas
- É comum o uso em variáveis de classe para definir **constantes**
 - PI, MAX_IDADE,...

Strings

- Você deve ter percebido no curso de Java básico que não existe o **tipo primitivo String** em Java
 - Em verdade, String's em Java são objetos!
 - A API Java possui uma classe chamada **String**

```
public class UsaString {  
    public static void main(String[] args){  
        String s1 = "Maria";  
        String s2 = new String("Maria");  
  
        if( s1 == s2 ) //falso  
        if( s1.equals(s2) ) // verdadeiro  
  
        System.out.print( s1 + " da Silva" );  
    }  
}
```

Arrays

- Também são objetos e armazenam elementos de um determinado tipo em particular
- Têm tamanho fixo depois de criados
- É possível declarar dois tipos de arrays:
 - ① **Primitivos**: armazenam tipos primitivos nas células
 - ② **Referência**: armazenam referências (**nunca objetos inteiros**) em cada célula

Arrays

```
public class ArrayPrimitivo {  
    public static void main(...) {  
        int[] vi = {55,66,77};  
        double[] vd = new double[7];  
  
        vd[6] = 99.5;  
        vi[2] = 88;  
    }  
}
```

```
public class ArrayReferencia {  
    public static void main(...) {  
        Livro[] vl = new Livro[3];  
  
        vl[0] = new Livro();  
        vl[0].setTitulo("O Sol");  
    }  
}
```

Boas práticas ao escrever classes

- Use e abuse dos espaços
 - Endente com um tab (4 espaços) os membros de uma classe
- A ordem dos membros não é importante, mas melhora a legibilidade do código
 - Mantenha os membros do mesmo tipo juntos (não misture métodos de classe com métodos de instância)
 - Declare os atributos antes ou depois dos métodos (não misture métodos com construtores ou variáveis)
 - Mantenha os construtores juntos, de preferência, bem no início da classe após os atributos

Exercícios

- Tentem codificar os exemplos mostrados nestes slides e verifique pontos de dúvidas
- **Práticas disponíveis no site da disciplina**

DÚVIDAS?