

Universidade Federal da Paraíba – Campus I
Centro de Informática
Departamento de Sistemas e Computação

Métodos e Projeto de Software
Material 6: Revisão de Conceitos
Básicos de O.O– Parte 1*

*Baseado no material de Helder da Rocha

Prof. Raoni Kulesza
raoni@ci.ufpb.br

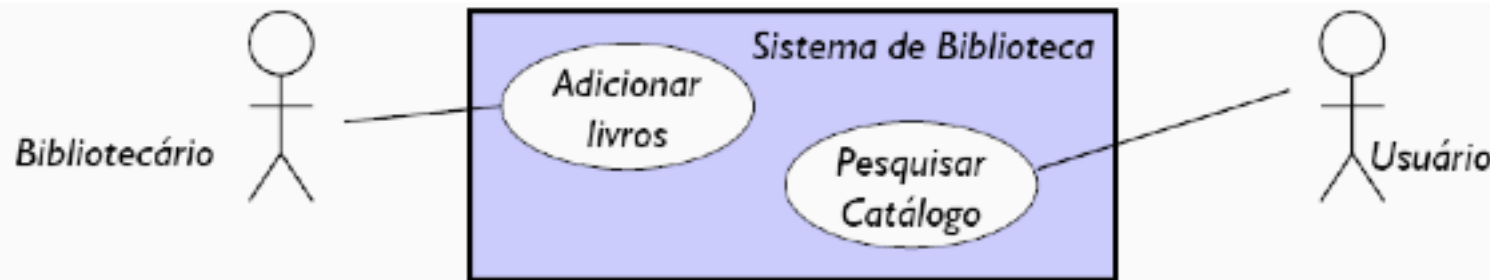


Programação estruturada vs POO

- 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)

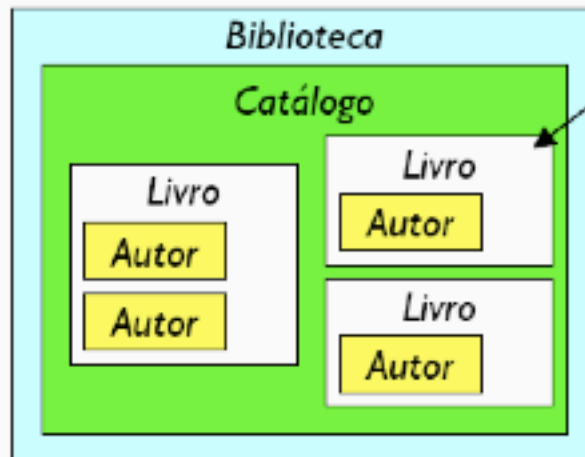


Análise OO (i) e Análise procedural (ii)



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

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

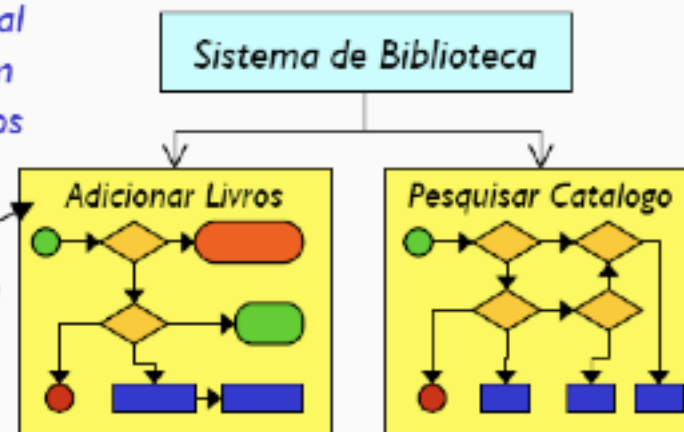


Lógica procedural encapsulada em objetos pequenos

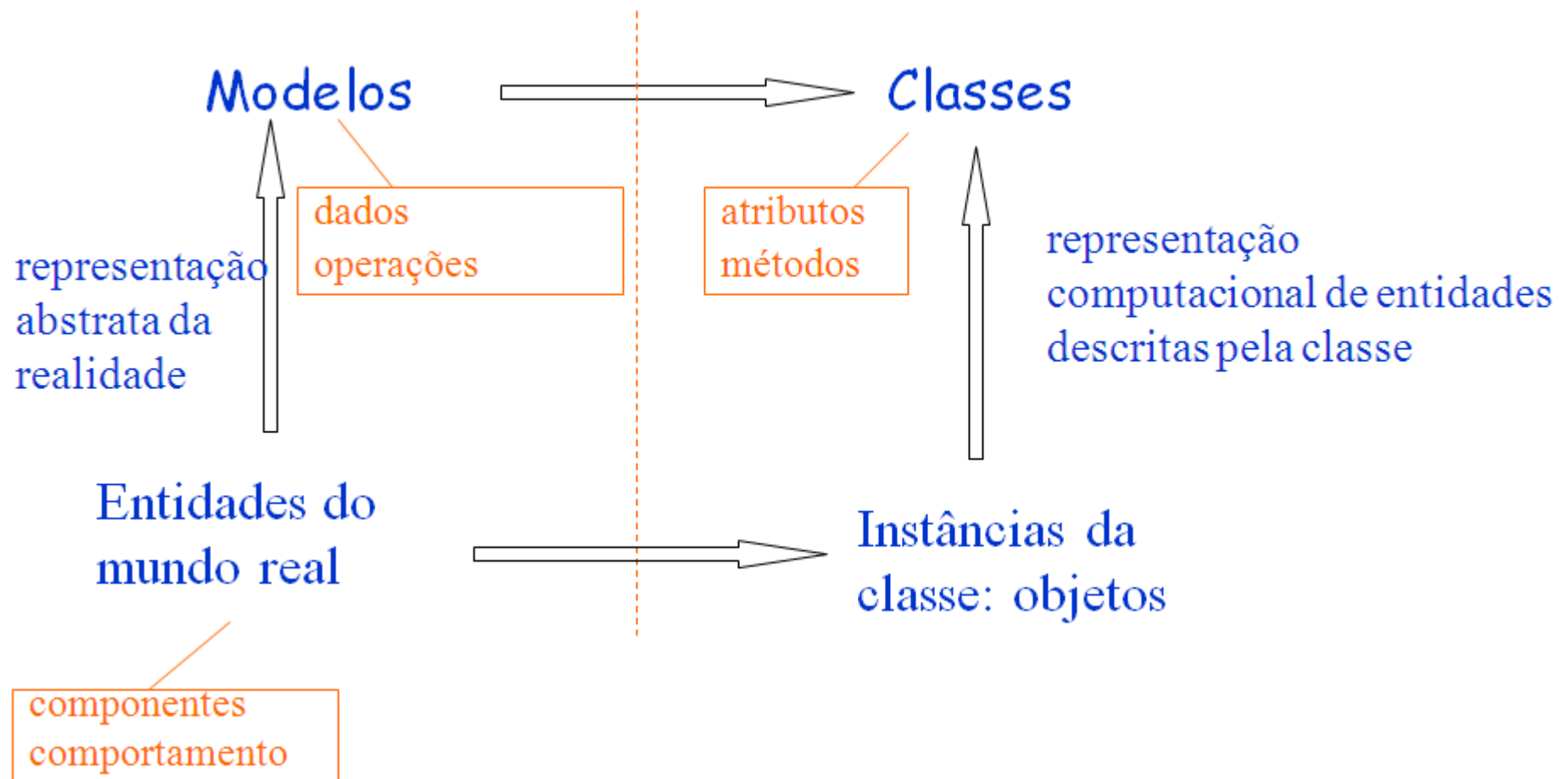
Lógica exposta e espalhada por todo o sistema

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

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

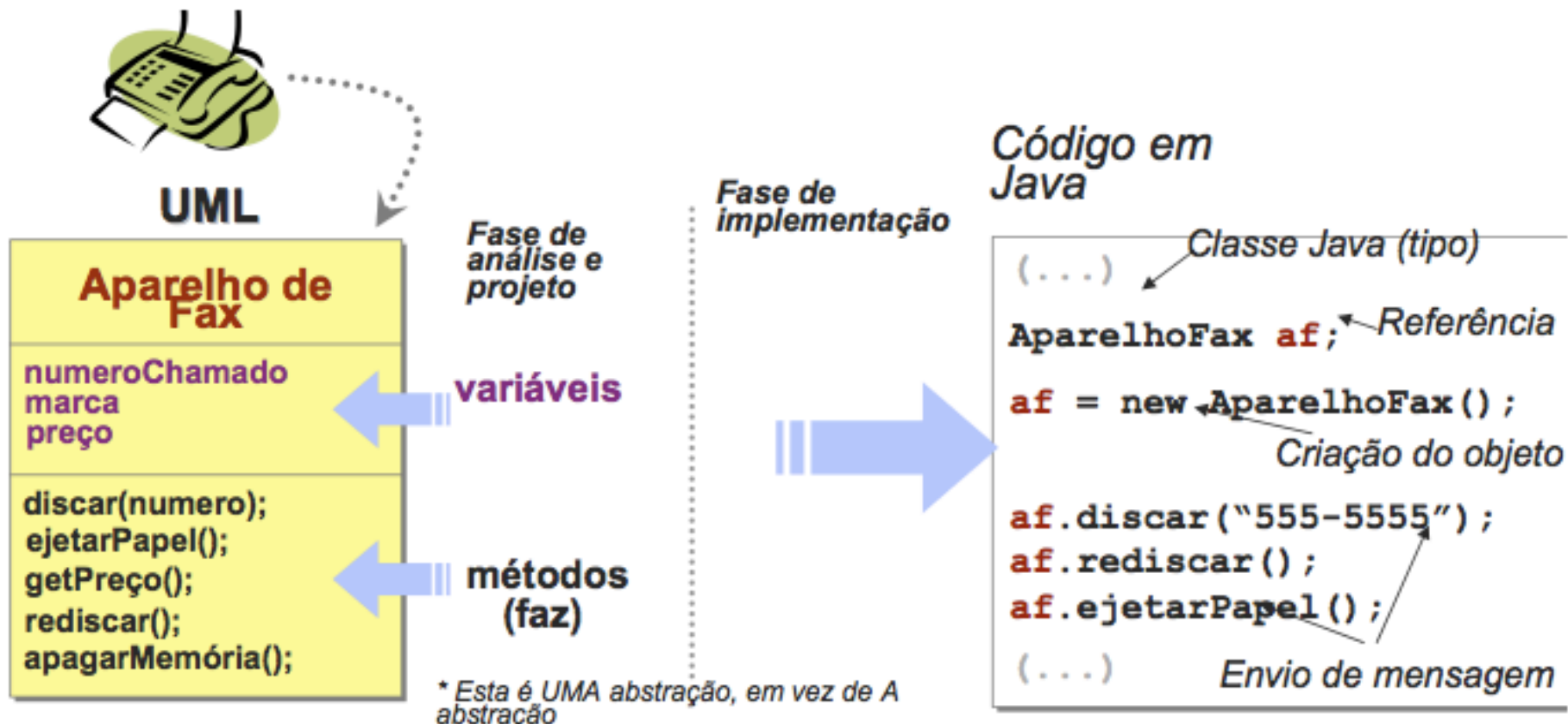


Análise Orientada a Objetos



Como fazer a modelagem inicial?

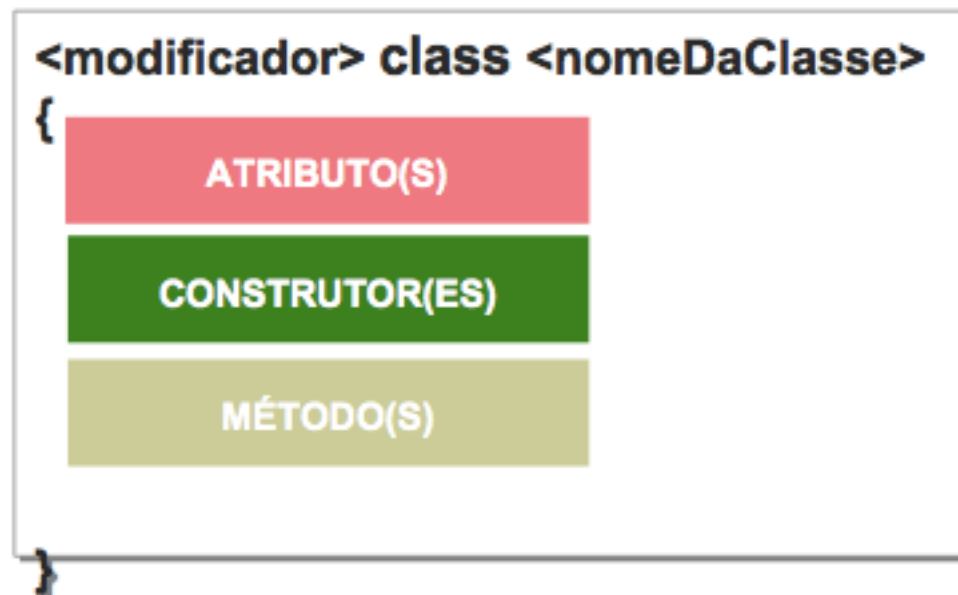
- **Classes** descrevem:
 - Os dados que compõem os objetos (**variáveis**)
 - Procedimentos que o objeto poderá executar (**métodos**)



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*

```
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



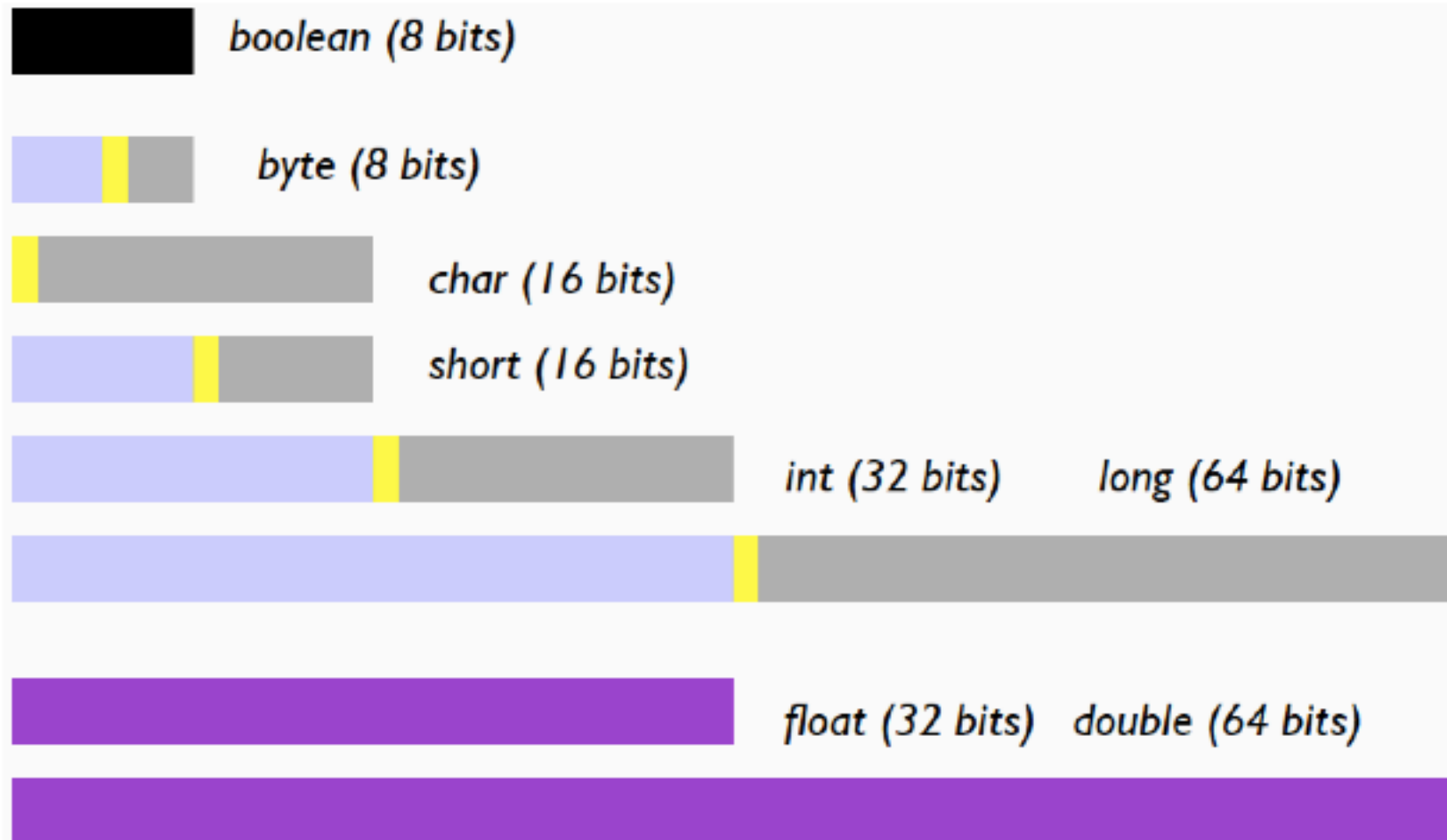
Atributos

- 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



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

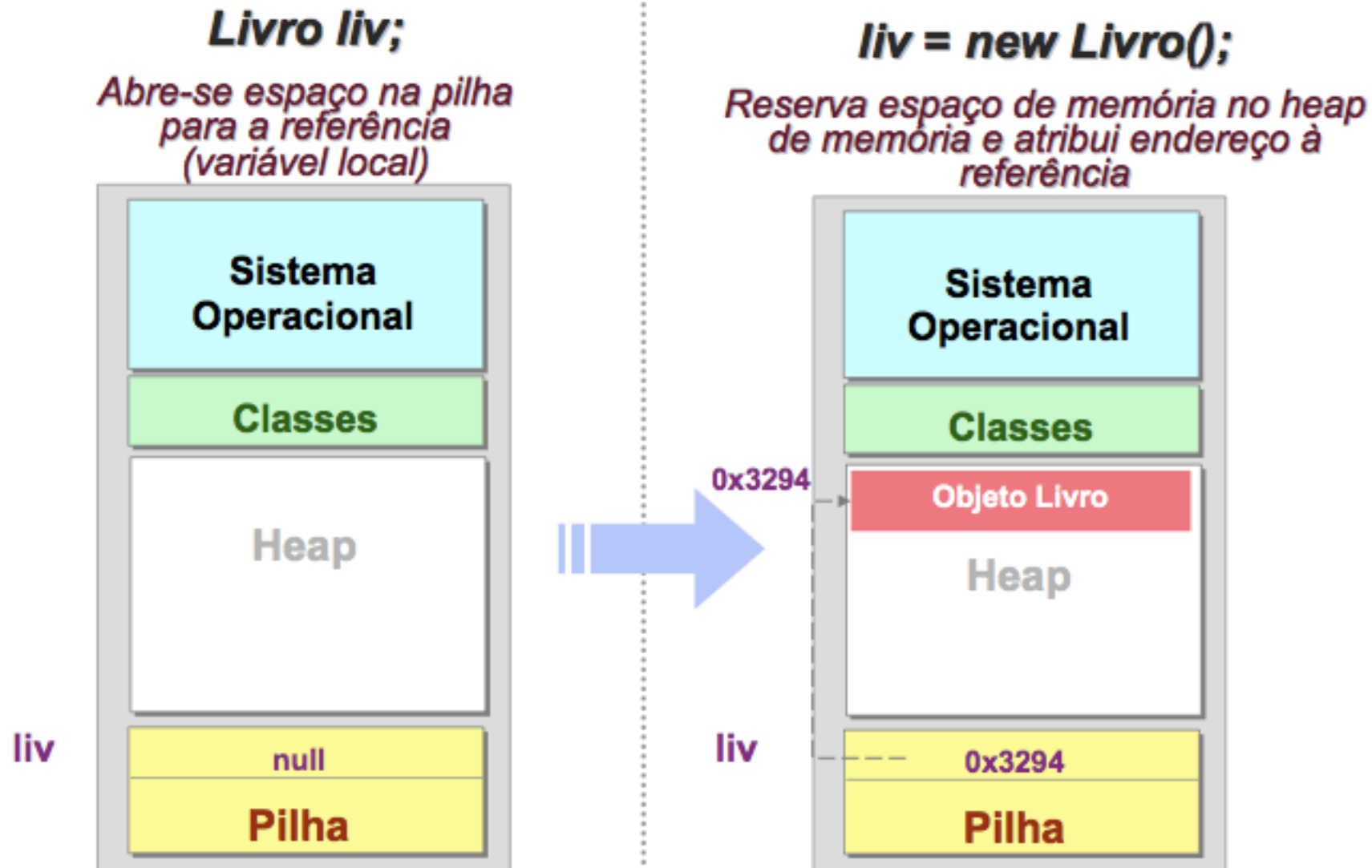
```
public class UsaLivro {  
    public static void main(String[] args){  
        Livro liv;  
  
        liv = new Livro();  
        liv.setTitulo("O Sol");  
        liv.setPreco(69.90);  
  
        liv = null; //coletor lixo  
    }  
}
```



null



Tipos de Referência: Detalhes



Tipos primitivos x referência

■ Variáveis de tipos primitivos

Pilha após linha 2

letraPri	'a'
letraPri2	'a'

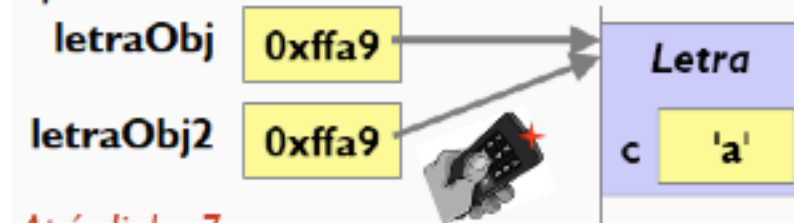
Pilha após linha 3

letraPri	'b'
letraPri2	'a'

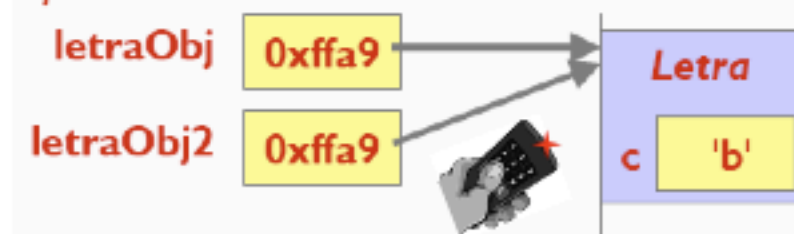
```
(...)  
1: char letraPri = 'a';  
2: char letraPri2 = letraPri;  
3: letraPri = 'b';  
(...)
```

■ 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:
 - Interface: métodos declarados (visão externa)
 - 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()

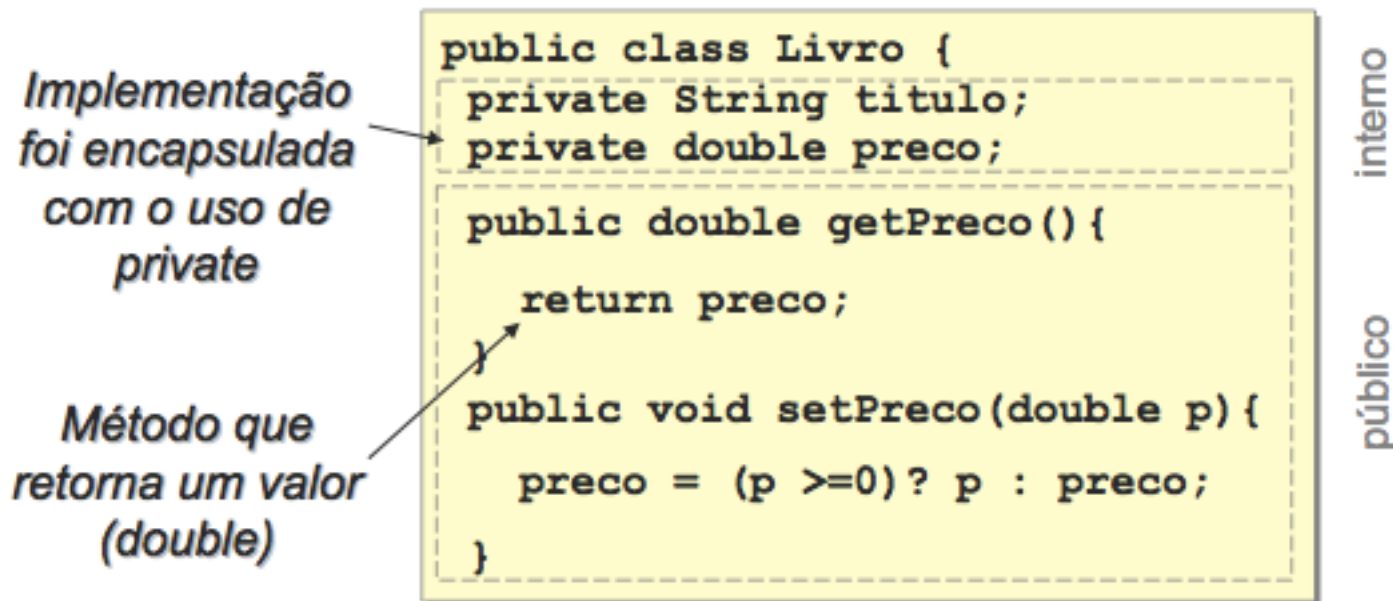
alterarHora()

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

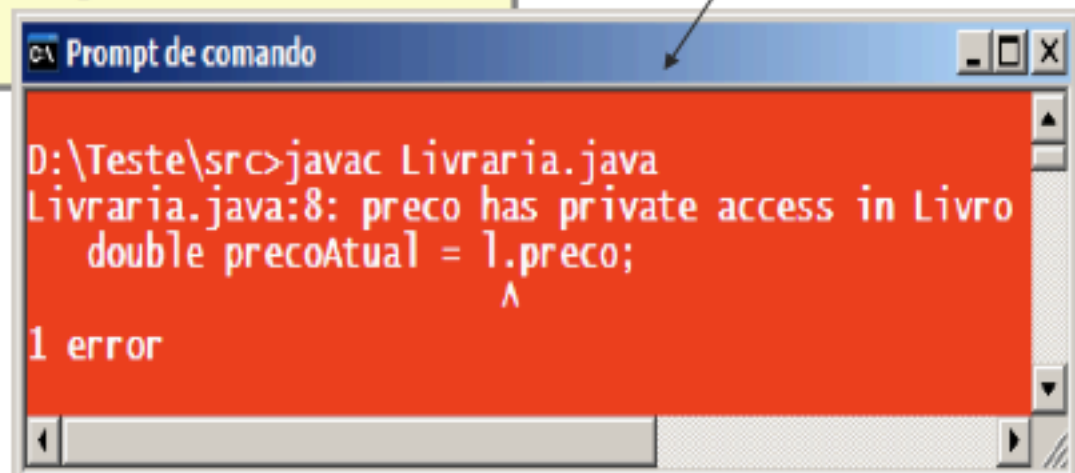


Cenário de Utilização

- 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



Prompt de comando

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

Acesso proibido!



Construtores

- 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**
- **Veremos mais detalhes na próxima aula !**

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

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



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*

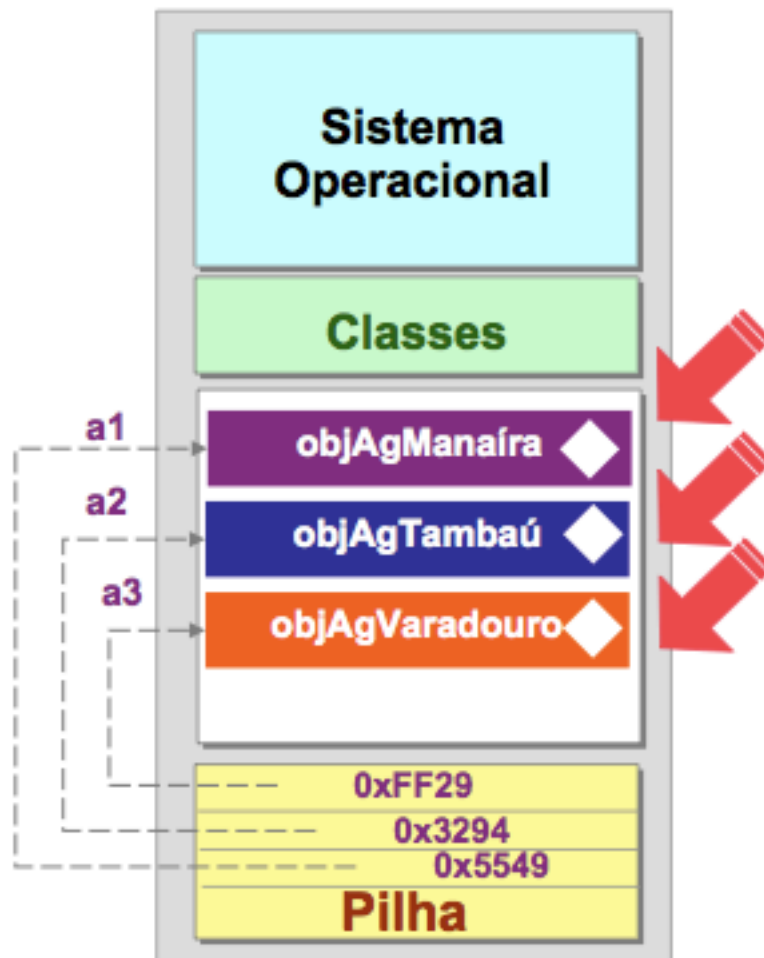
```
public class Agencia {  
    private String endereco;  
    private int nome;  
    public static double juros = 0.4;  
  
    public String getEndereco() {...}  
    public void ligarAlarme() {...}  
}
```



Graficamente

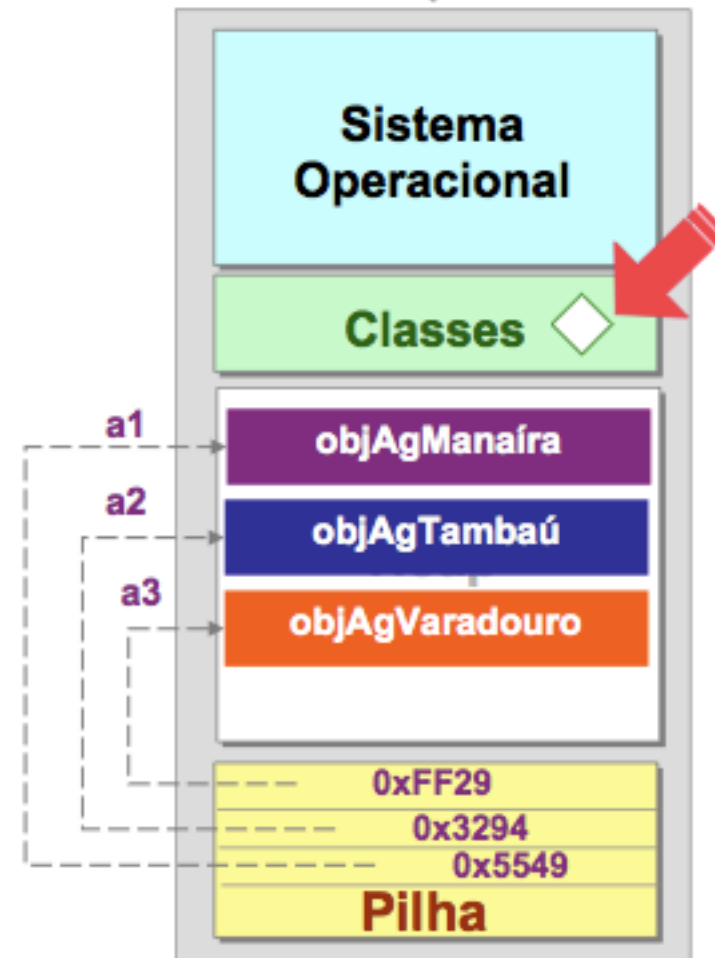
1ª tentativa

*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 (Vetores)

- 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
 - **Primitivos:** armazenam tipos primitivos nas células
 - **Referência:** armazenam referências (**nunca objetos inteiros**) em cada célula



Arrays (Vetores)

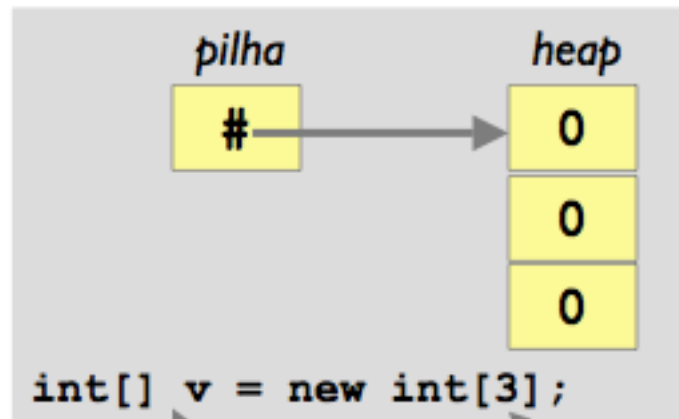
```
public class ArrayPrimitivo {  
    public static void main(...) {  
        int[] vi = {55,66,77};  
        double[] vd = new double[7];  
  
        vd[6] = 99.5;  
        vi[2] = 88;  
    }  
}
```

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



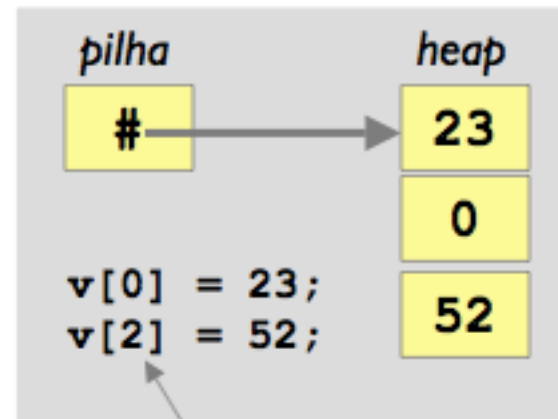
Arrays (Vetores)

■ De tipos primitivos



v é objeto do tipo (int[])

cria um vetor

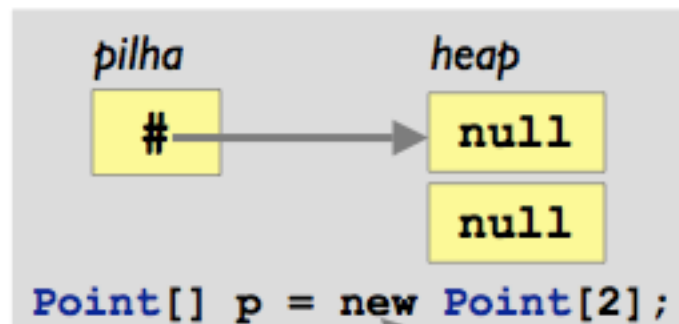


inicialização dos elementos

```
class Point {  
    public int x;  
    public int y;  
}
```

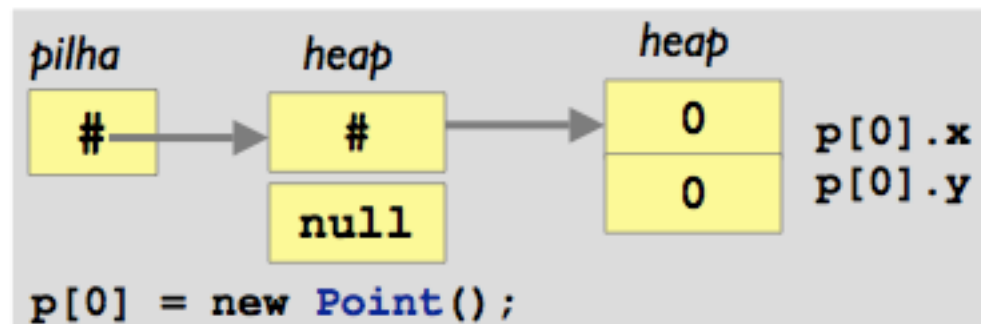
Point
+x: int
+y: int

■ De objetos (*Point* é uma classe, com membros x e y, inteiros)



p é objeto do tipo (Point[])

cria um vetor



cria um objeto Point



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 na “compilação”, 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
- Utilize uma convenção de código
 - <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>
 - <https://google-styleguide.googlecode.com/svn/trunk/javaguide.html>



Dúvidas?

raoni@ci.ufpb.br

