

# PROGRAMAÇÃO ORIENTADA A OBJETOS

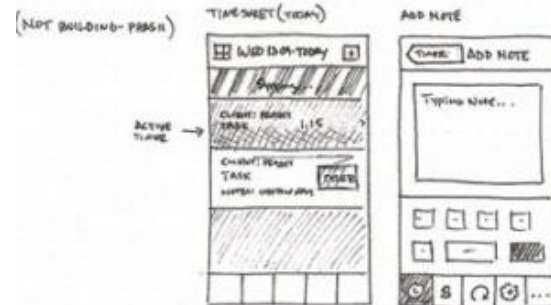
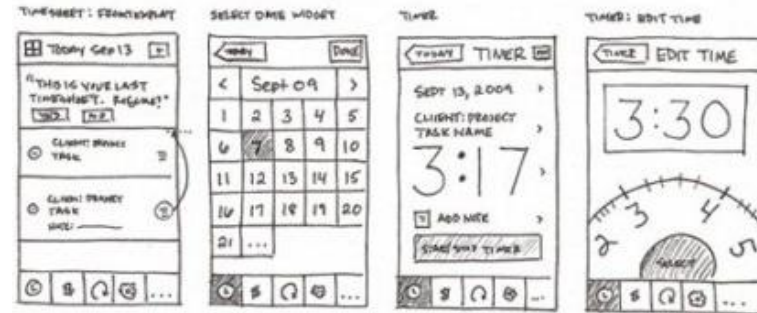
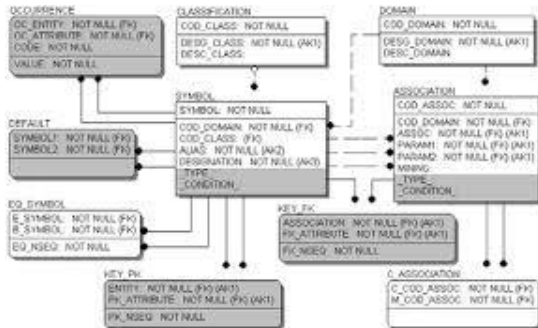
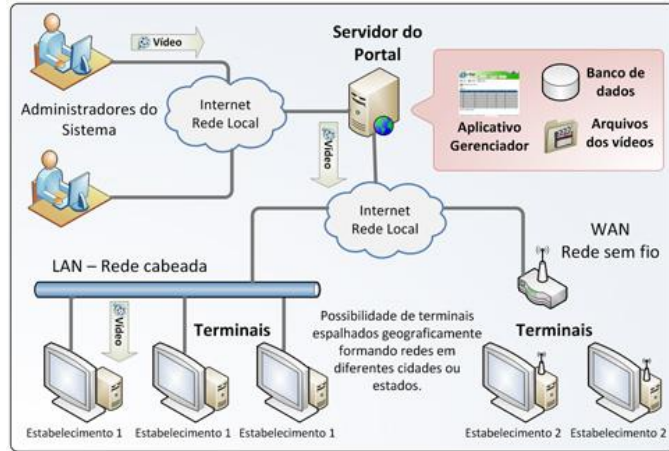
## Parte 01

3º período

Prof. Marcello Thiry <marcello.thiry@gmail.com>

# O que é “Abstração”

2





# O que é “Abstração”

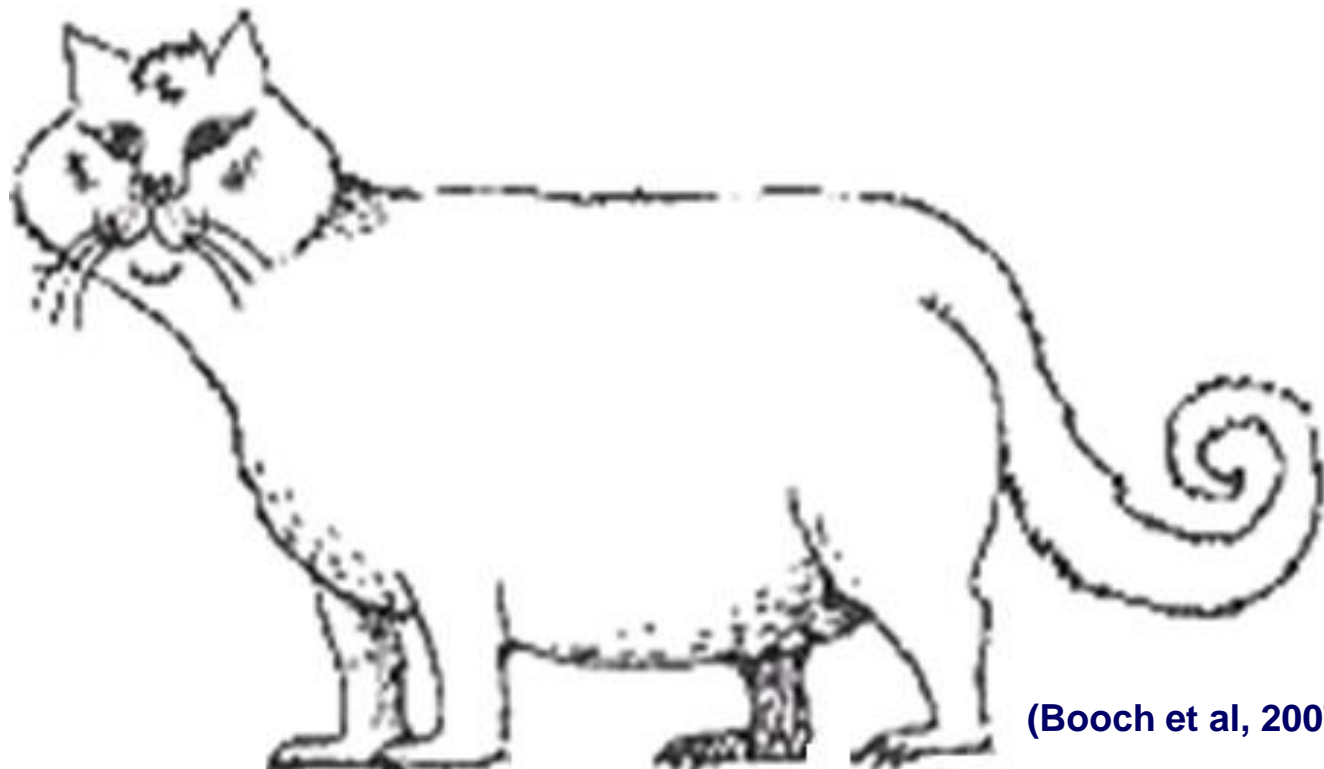
3

- Processo de **identificar os aspectos essenciais** de um contexto qualquer, ignorando características menos importantes ou acidentais
- “**Uma abstração**” é o resultado deste processo

# O que você está vendo?



4

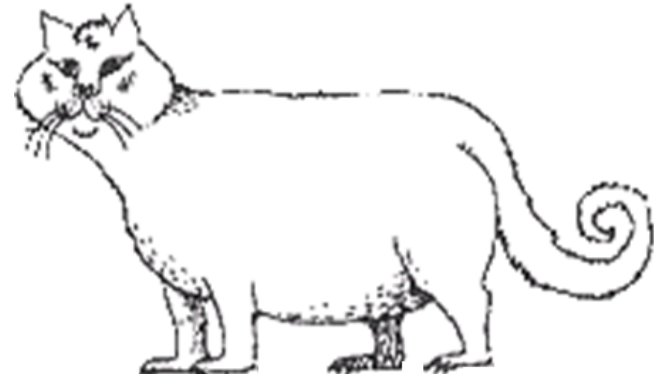


(Booch et al, 2007)

# Diferentes níveis de abstração

5

- Animal
  - ▣ Mamífero
    - Gato
      - Gato Persa

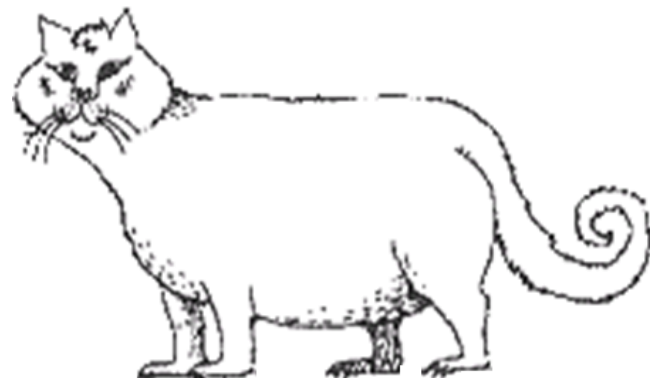


(Booch et al, 2007)

# Existem visões diferentes?

6

- A seleção de quais aspectos são essenciais **depende do observador e do fenômeno observado**

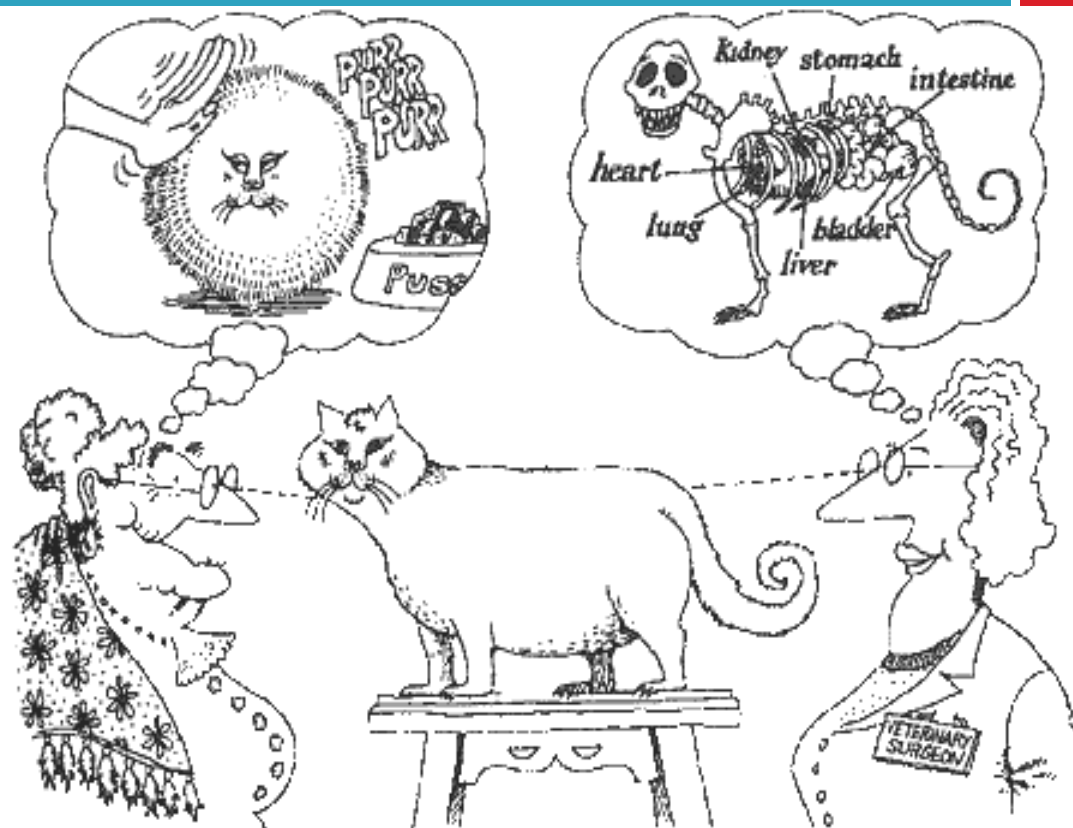


(Booch et al, 2007)

# Visões diferentes

□ Diferentes  
**observadores**

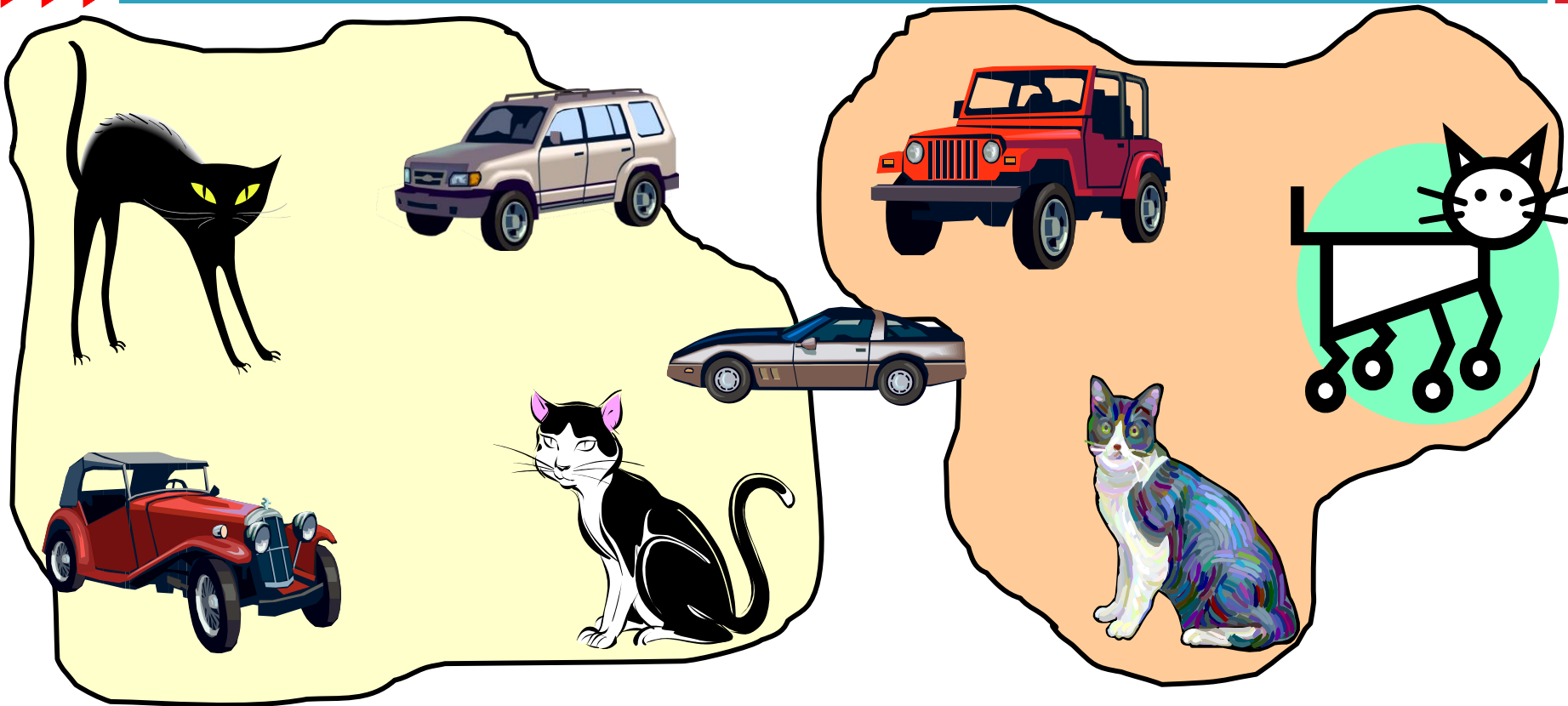
□ Diferentes  
**necessidades**



(Booch et al, 2007)

# Como você classificaria estas “coisas”?

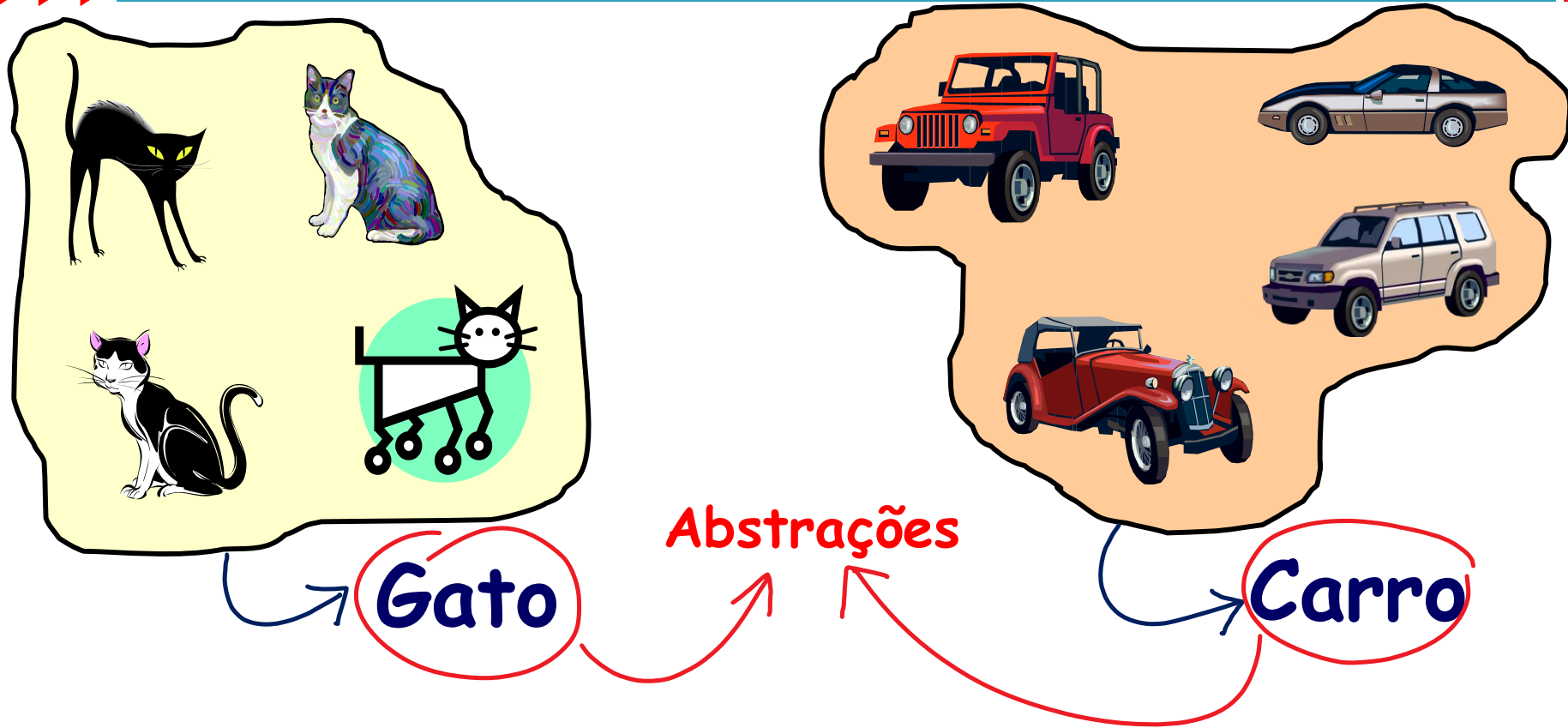
8





# Classificar é uma forma de abstração

9



# Critérios para a classificação

10

❑ O que você observou em cada “coisa”?

❑ Similaridades?

- Conjunto de características comuns
- Comportamento comum
- Mas, cada “coisa” é um indivíduo dentro do grupo, certo?



❑ Adotaremos o termo “**objeto**” para “coisa”



# O que é um “Objeto”

11

- É uma **entidade** com **estado** e **comportamento** específico
- Frequentemente, objetos de software modelam objetos do mundo real



# Objeto

12

## □ Estado

- ▣ Martelo Tipo I
- ▣ Cabo de nogueira
- ▣ Cabeça de aço

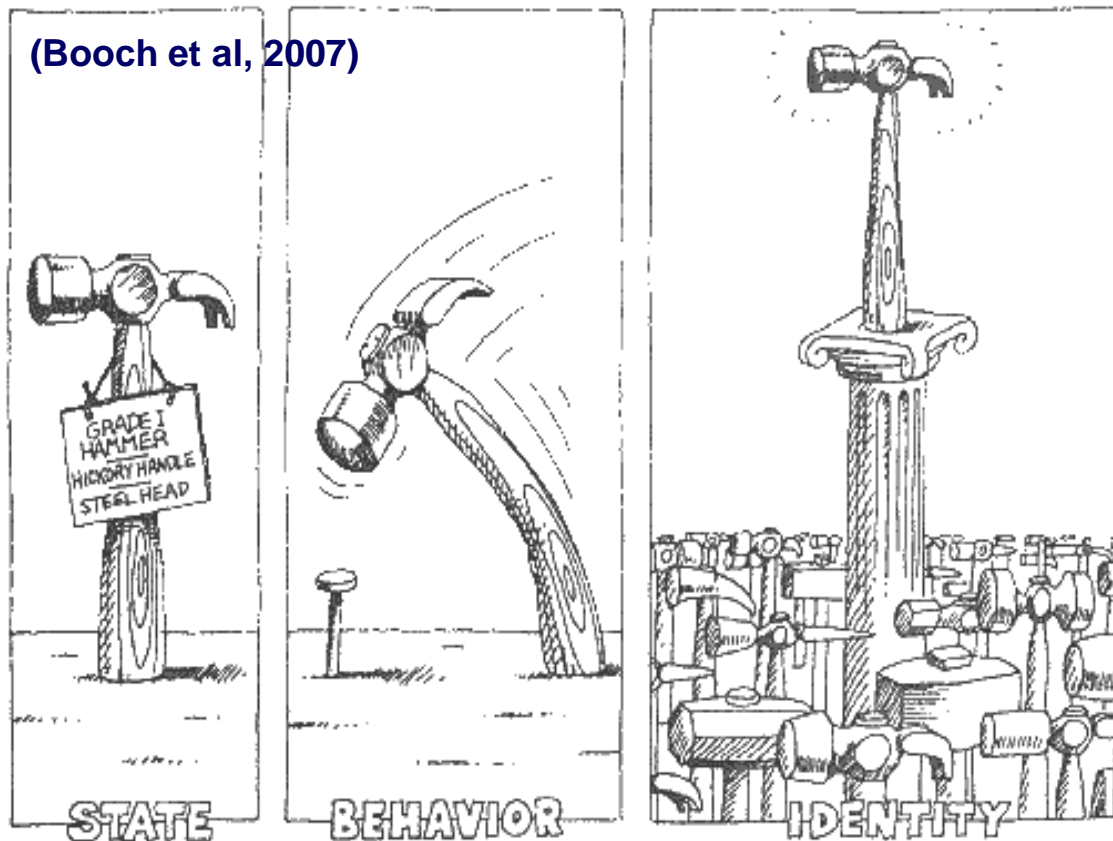
## □ Comportamento

- ▣ Martelar

## □ Identidade

- ▣ Meu martelo

(Booch et al, 2007)





# Estado de um Objeto

13

- Um objeto é caracterizado por um conjunto de **atributos**
- Atributos são **características** (propriedades) presentes em todos os objetos de uma mesma classe

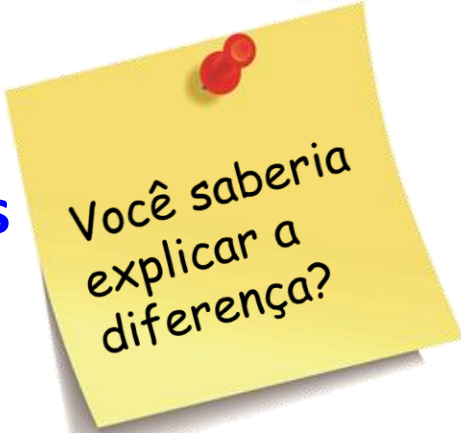


# Estado de um Objeto

14

- O **conjunto dos valores** de cada atributo em um determinado momento representa o **estado de um objeto**

- Não confundir atributos com variáveis locais



Você saberia explicar a diferença?

# Atributos e variáveis locais

15

- ❑ Um **atributo** é uma **característica relevante** presente em um objeto durante toda a vida deste objeto
  - ▣ Veja se faz sentido falar de um objeto sem aquele atributo
- ❑ Variáveis temporárias não devem ser declaradas como atributos
  - ▣ Ex: variáveis que controlam laços; variáveis que guardam valores intermediários em cálculos.

# Estado de um Objeto

16

## □ Objeto “Meu Ventilador”

- Número de pás: 3
- Tipo de pá: **MADEIRA**
- Número de velocidades: 3
- Cor: **MOGNO**
- Tem exaustor: **SIM**
- Tem lustre: **SIM**



**Atributos**



# Estado de um Objeto

17

## ❑ Objeto “Meu Ventilador”

- ❑ Número de pás: 3
- ❑ Tipo de pá: MADEIRA
- ❑ Número de velocidades: 3
- ❑ Cor: MOGNO
- ❑ Tem exaustor: SIM
- ❑ Tem lustre: SIM



Estado atual

# Estado de outro Objeto

18

## ❑ Objeto “Outro Ventilador”

- ❑ Número de pás: 2
- ❑ Tipo de pá: **PLÁSTICO**
- ❑ Número de velocidades: 4
- ❑ Cor: **VERDE**
- ❑ Tem exaustor: **NÃO**
- ❑ Tem lustre: **SIM**



# Comportamento de um Objeto

19

- ▶▶▶
- ❑ Como pedir ao objeto que ele faça alguma ação?
- ❑ Quais as ações sobre um ventilador?
  - ▣ Alterar o número de pás
  - ▣ Obter o número de pás?
  - ▣ Alterar a cor
  - ▣ Obter a cor?
  - ▣ ...



# Comportamento de um Objeto

20

- ❑ Existem outras ações?
  - ❑ Ligar / desligar modo ventilador
  - ❑ Ligar / desligar modo exaustor
  - ❑ Acender a luz
  - ❑ Desligar a luz
  - ❑ Aumentar velocidade
  - ❑ ...



# Comportamento de um Objeto

21

- ❑ Mais ações?
  - ❑ Está ligado ou não?
  - ❑ Em qual modo está ligado?
  - ❑ A luz está acesa?
  - ❑ Qual a velocidade atual?
  - ❑ ...



# Comportamento de um Objeto

22

- Como você identificou o comportamento?
  - ▣ Qual a relação com o processo de abstração?
  - ▣ Qual o seu ponto de vista?
    - A ideia seria desenvolver um sistema de controle de estoque?
    - A ideia seria desenvolver um sistema de controle automático dos ventiladores em um prédio inteligente?
    - ??
- ▣ **O ponto de vista (necessidades) afeta a abstração!!**



# Comportamento de um Objeto

23

- O comportamento de um objeto é representado por um conjunto de operações
- ▣ Operações representam as **responsabilidades da classe**, determinando **o que os objetos podem fazer** (ações)

# Identidade de um Objeto

24



SN#123345



SN#453425



SN#434247



SN#223450



SN#112112



SN#332232



SN#098934



SN#654874



SN#457778



SN#988524



# Identidade de um Objeto

25

- ❑ Objetos podem ter um mesmo Estado...
- ❑ **Mas, eles ainda são objetos diferentes!!**





# O que é uma “Classe”

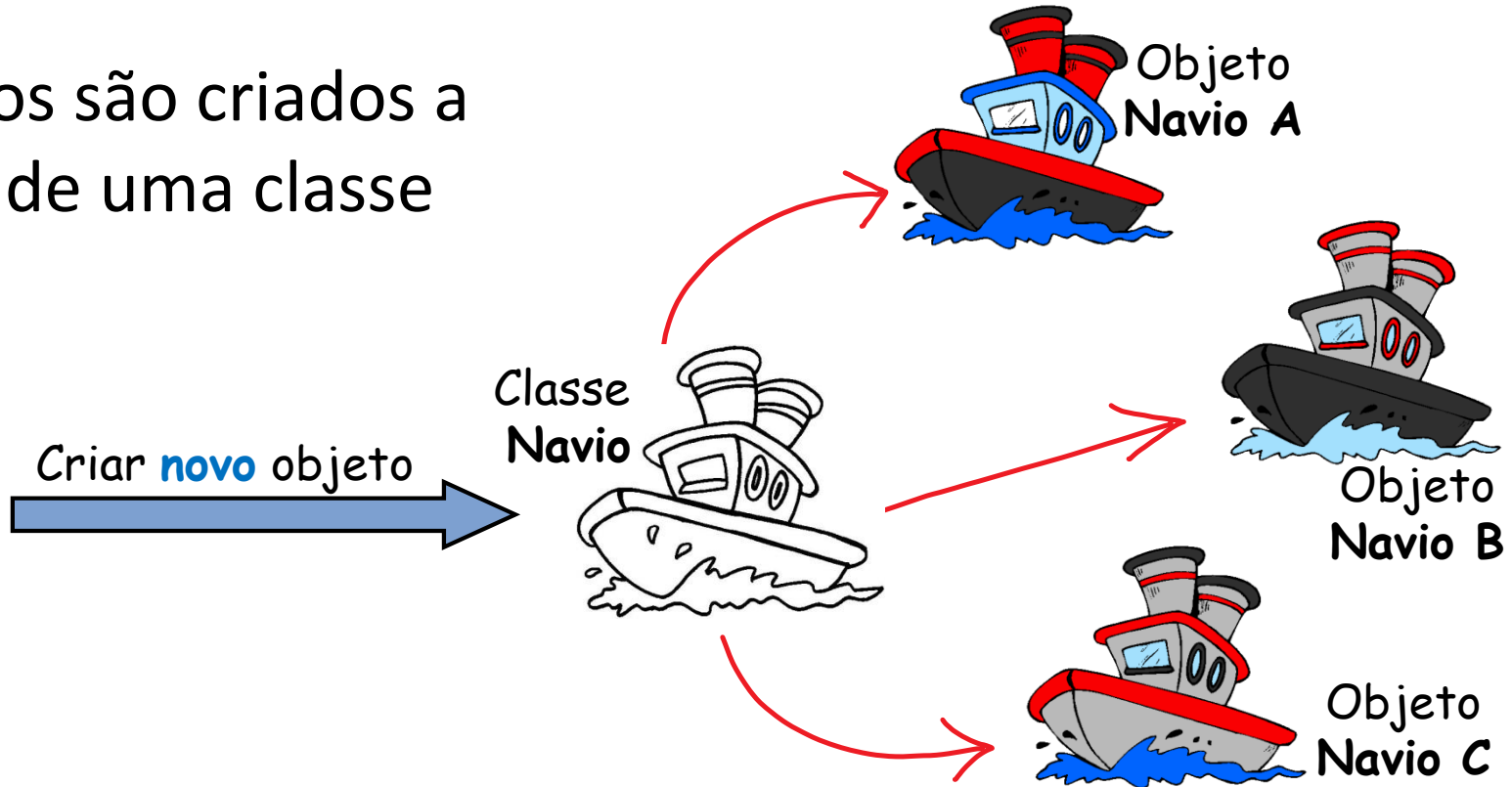
26

- É um molde a partir do qual objetos são criados
- Define **dados (atributos) e comportamentos (operações) comuns** a todos os objetos criados a partir dela



# Instanciando objetos

- Objetos são criados a partir de uma classe



# Instanciando objetos

28

- Cada objeto é uma instância de sua classe
- Criação = instanciação



# Instanciando objetos

29

- Note que você deve pedir para a classe “**instanciar**” um novo objeto
- Você entende o motivo?



# Instanciando objetos

30



# Destruindo objetos

31

- Objetos podem ser destruídos

**Destruir** objeto



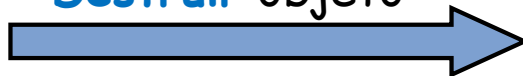
Objeto Navio A

**Destruir** objeto



Objeto Navio B

**Destruir** objeto

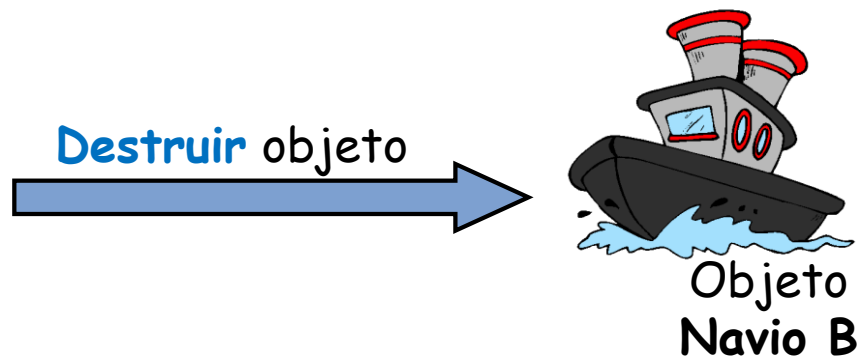


Objeto Navio C

# Destruindo objetos

32

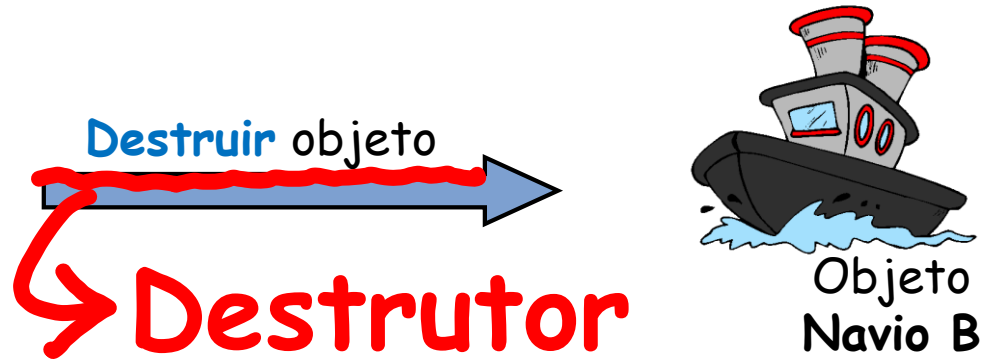
- Note que o pedido é feito para o objeto e não para a sua classe
- Você entende o motivo?





# Destruindo objetos

33





# Mudança de paradigma

34

## Unidade de Código

**Dados**

**Operações para  
manipular os dados**

## Objeto

**Atributos (dados)**

**Operações  
para  
manipular  
os dados**

# Encapsulamento

35

## Objeto

Atributos (dados)

Operações  
para  
manipular  
os dados

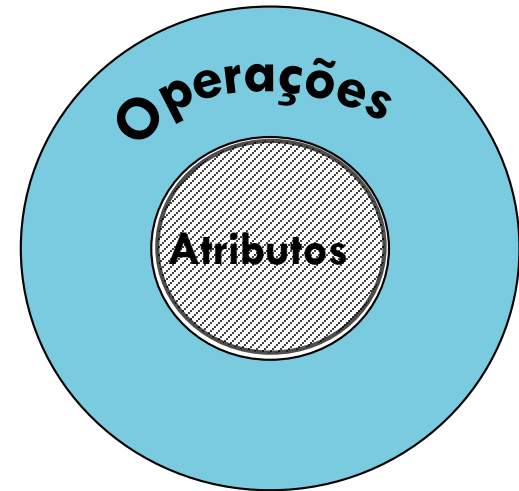
**Atributos e operações estão encapsulados em uma mesma estrutura**



# Encapsulamento

36

- ❑ Mecanismo para agrupar os atributos com as respectivas operações que manipulam estes atributos



# Acesso indireto aos atributos do objeto

37

## Objeto

Atributos (dados)

Operações  
para  
manipular  
os dados

operação 1

operação 2

operação N

## Programas usuários

```
01101 Private Function CleanUpLine(ByVal sLine As String) As String
01102 Dim iQuoteCount As Long
01103 Dim iLocat As Long
01104 Dim sChar As String
01105 Dim sPrevChar As String
01106
01107 ' Starts with Rem if it is a comment
01108 stime = Trim(stime)
01109 If Left(stime, 3) = "Rem" Then
01110 CleanUpLine = ""
01111 Exit Function
01112 End If
01113
01114 ' Starts with ' it is a comment
01115 If Left(stime, 1) = "'" Then
01116 CleanUpLine = ""
01117 Exit Function
01118 End If
01119
01120 ' Contains ' any and in a comment, so test if it is a comment or in the
01121 body of a string
01122 If Instr(stime, "'") > 0 Then
01123 sPrevChar = ""
01124 iQuoteCount = 0
01125
01126 For iLocat = 1 To Len(stime)
01127 sChar = Mid(stime, iLocat, 1)
01128
01129 ' If we found ' then an even number of ' characters in front
01130 ' means it is the
01131 ' part of a string
01132 If sChar = "'" Then
01133 iQuoteCount = iQuoteCount + 1
01134 sPrevChar = sChar
01135 Exit For
01136 End If
01137
01138 ElseIf sChar = " "
01139 iQuoteCount = iQuoteCount + 1
01140 End If
01141 sPrevChar = sChar
01142 Next iLocat
01143
01144 CleanUpLine = stime
01145 End Function
```

```
01101 Private Function CleanUpLine(ByVal sLine As String) As String
01102 Dim iQuoteCount As Long
01103 Dim iLocat As Long
01104 Dim sChar As String
01105 Dim sPrevChar As String
01106
01107 ' Starts with Rem if it is a comment
01108 stime = Trim(stime)
01109 If Left(stime, 3) = "Rem" Then
01110 CleanUpLine = ""
01111 Exit Function
01112 End If
01113
01114 ' Starts with ' it is a comment
01115 If Left(stime, 1) = "'" Then
01116 CleanUpLine = ""
01117 Exit Function
01118 End If
01119
01120 ' Contains ' any and in a comment, so test if it is a comment or in the
01121 body of a string
01122 If Instr(stime, "'") > 0 Then
01123 sPrevChar = ""
01124 iQuoteCount = 0
01125
01126 For iLocat = 1 To Len(stime)
01127 sChar = Mid(stime, iLocat, 1)
01128
01129 ' If we found ' then an even number of ' characters in front
01130 ' means it is the start of a comment, and odd number means it is
01131 ' part of a string
01132 If sChar = "'" Then
01133 iQuoteCount = iQuoteCount + 1
01134 sPrevChar = sChar
01135 Exit For
01136 End If
01137
01138 ElseIf sChar = " "
01139 iQuoteCount = iQuoteCount + 1
01140 End If
01141 sPrevChar = sChar
01142 Next iLocat
01143
01144 CleanUpLine = stime
01145 End Function
```



# E as variáveis globais?

38



# Atributos devem ser protegidos!!

39



- O que acontece se qualquer um tiver acesso a sua carteira exatamente neste momento?





# Ocultação da informação\*

40

- ❑ **Esconder o estado** do objeto
  - ▣ As operações são a única forma de acessar ou modificar um objeto
  - ▣ **Você entendeu porque não é necessário visualizar diretamente os atributos (estado) do objeto?**



\*Information hiding



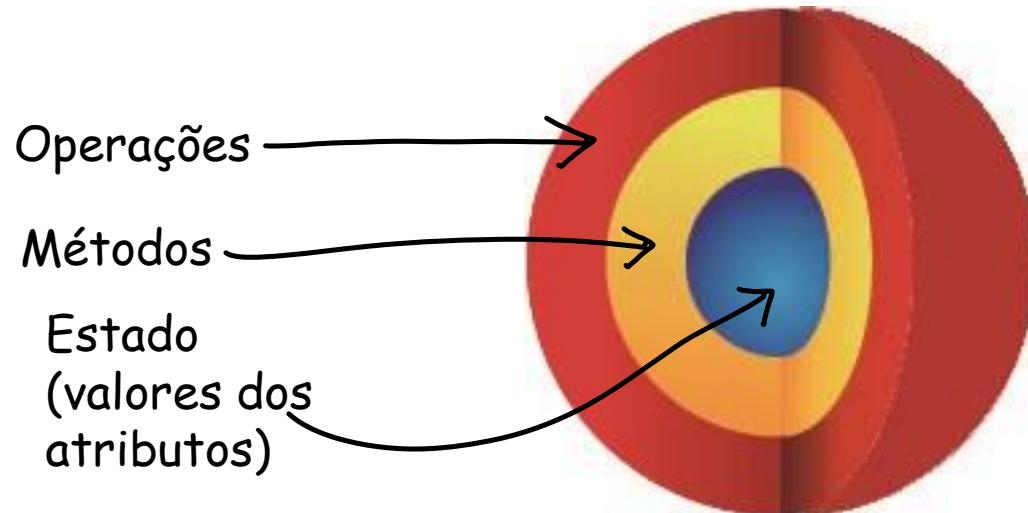


# Encapsulamento

41

□ Permite **esconder a implementação das operações**

▣ A implementação de uma operação é denominada **método**



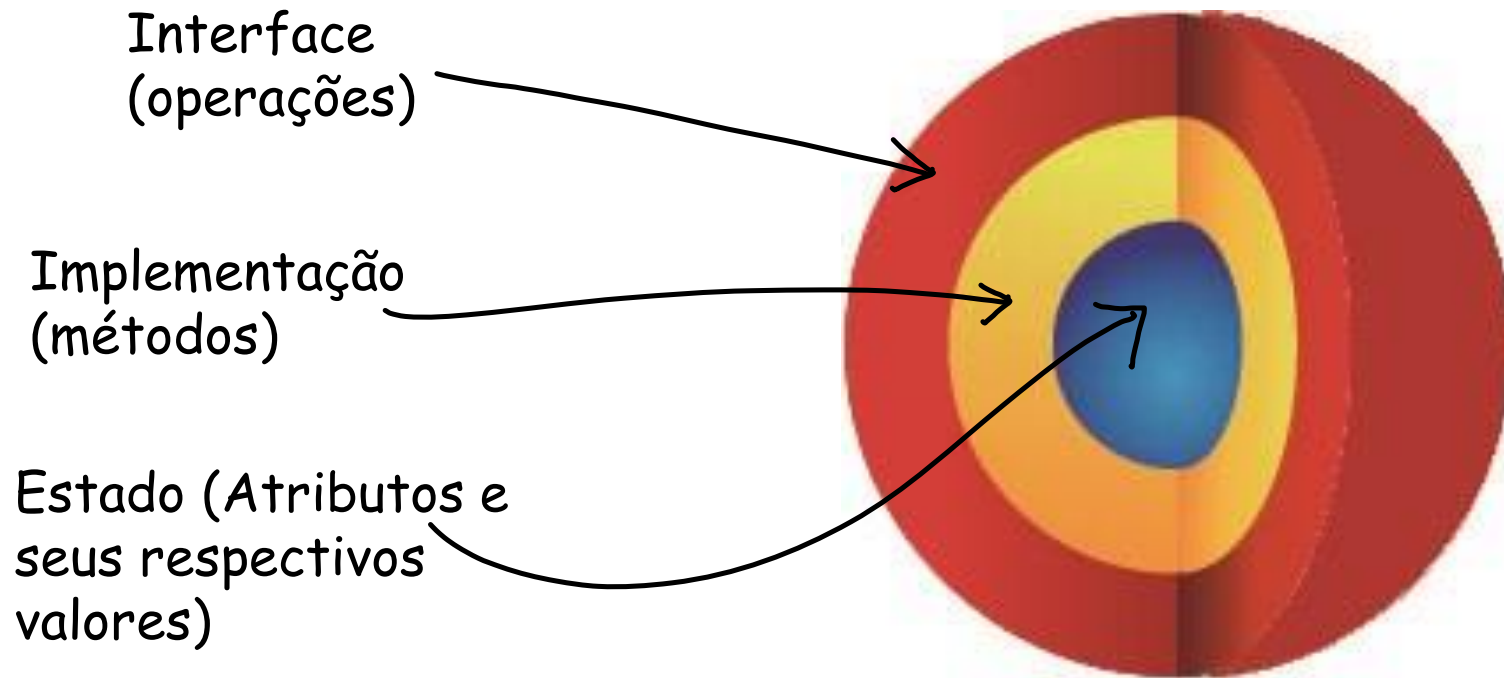
# Encapsulamento e Interface

42

- Toda classe implementa (realiza) uma **interface**
  - ▣ A **interface** define o conjunto de operações visíveis (públicas) de uma classe
    - Define quais operações podem ser **invocadas** nos objetos de uma determinada classe
  - ▣ Encapsulamento permite a separação entre a interface (operações) e a sua implementação (métodos)

# Encapsulamento e Interface

43



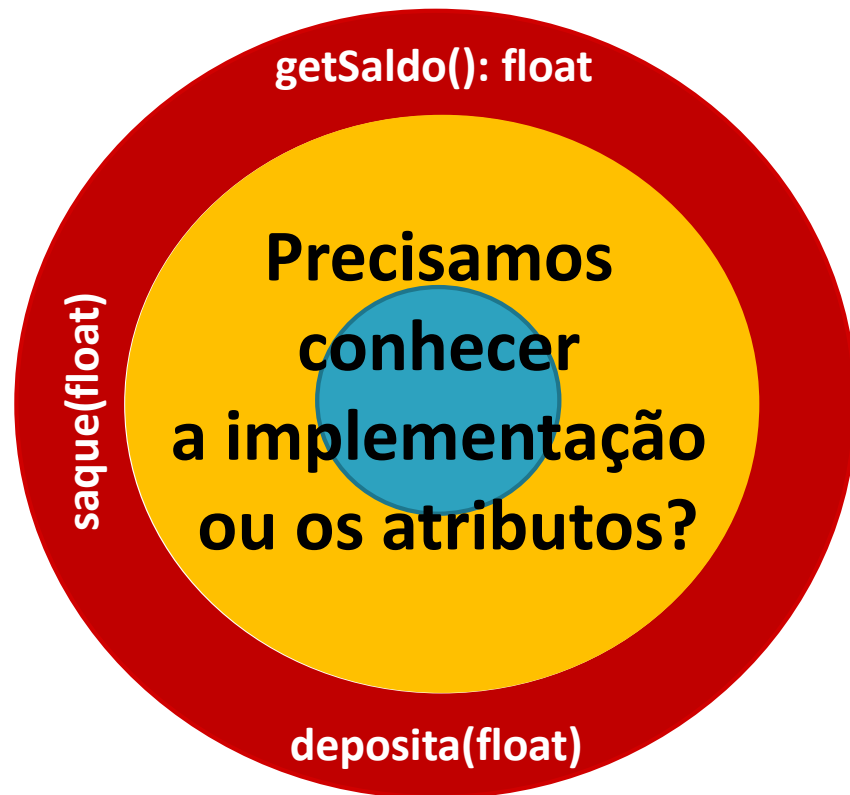
# Encapsulamento + Ocultamento da Informação

44

❑ Exemplo:

## ❑ ContaBancaria

- Como saber quantos reais temos na conta?
- Como colocar mais dinheiro na conta?
- Como retirar dinheiro da conta?



# Como nomear uma Classe

45

- O nome da classe deve estar relacionado com o **principal objetivo** desta classe
- ▣ Qual o conceito (abstração) que a classe representa?
  - Se a abstração foi bem conduzida, será fácil dar um nome a classe
  - Caso contrário, você deve rever a abstração

# Como nomear uma Classe

46

- Aplicar o **Princípio da Responsabilidade Única**:
  - ▣ Uma classe deve ter uma única responsabilidade
  - ▣ Esta responsabilidade deve ser totalmente encapsulada pela classe
  - ▣ Todas as operações devem estar fortemente alinhadas a esta responsabilidade

# Como nomear uma Classe

47

- ❑ Utilizar um **substantivo** que represente claramente a abstração:
  - ▣ Professor, Cliente, Automovel, Produto, ...
  - ▣ NotaFiscal, Contracheque,
  - ▣ Pilha, Lista, Fila...
- ❑ Tipicamente, o nome da classe é expresso no singular
  - ▣ Utilize a relação “é um” / “é uma” dos objetos para a classe

# Como nomear uma Classe

48

□ Adote um estilo de escrita:

□ **UpperCamelCase**<sup>1</sup>

- Quando o nome da classe for composto por mais de uma palavra, elas são concatenadas diretamente
  - Não use underline/underscore para separar as palavras
- Cada palavra é iniciada com uma letra maiúscula
- Ex: ContaBancaria, Disciplina, Curso, FiguraGeometrica, ...

1. <http://pt.wikipedia.org/wiki/CamelCase>



# Como nomear uma Classe

49

- Considerar a métrica de **Coesão**<sup>1</sup>
  - ▣ Representa a relação existente entre as operações de uma determinada classe e de sua responsabilidade
  - ▣ Métrica **interna** a uma classe
  - ▣ O objetivo é manter **ALTA COESÃO**
    - Quanto maior for a coesão, melhor aplicado foi o **Princípio da Responsabilidade Única**

1. *Coesão é uma medida com origem na abordagem estruturada, relacionada ao Princípio da Modularidade, indicando o quanto os elementos de um módulo (dados, procedimentos e funções) fazem sentido juntos.*

# Como nomear um Atributo

50

## □ Adote um estilo de escrita:

### ▣ LowerCamelCase<sup>1</sup>

- Quando o nome do atributo for composto por mais de uma palavra, elas são concatenadas diretamente
  - Não use underline/underscore para separar as palavras
- A primeira palavra é iniciada com uma letra minúscula
  - As demais palavras são iniciada com uma letra maiúscula
- Ex: dataNascimento, nomeCompleto, enderecoResidencial...

1. <http://pt.wikipedia.org/wiki/CamelCase>

# Como nomear um Atributo

51

- ❑ Utilize nomes expressivos que tenham relação com o problema a ser resolvido
  - ▣ O nome deve mostrar claramente o seu objetivo
    - O que o atributo representa
    - Utilize o domínio do problema para identificar nomes adequados
  - ▣ Evite mnemônicos, mas cuidado com nomes muito longos
  - ▣ Lembre-se que os atributos serão tipicamente objetos de outras classes

# Como nomear uma Operação

52

- Adote um estilo de escrita
  - ▣ **LowerCamelCase**<sup>1</sup> (mesmo estilo usado nos atributos<sup>2</sup>)
  - ▣ Exemplos:
    - getSaldo
    - setEnderecoResidencial
    - transmitirArquivoContas
    - ...

1. <http://pt.wikipedia.org/wiki/CamelCase>

2. *Este estilo de escrita é geralmente adotado para a linguagem Java. Entretanto, outras linguagens podem adotar estilos diferentes. O mais importante é adotar um padrão de escrita.*

# Como nomear uma Operação

53

- ❑ Utilize nomes expressivos que tenham relação com a ação a ser realizada
  - ▣ Evite mnemônicos, mas cuidado com nomes muito longos
  - ▣ Utilize o domínio do problema para identificar nomes adequados
    - Isso vale tanto para a operação quanto para os parâmetros
  - ▣ Evite muitos parâmetros
    - Mantenha legibilidade na **assinatura da operação**

# Assinatura de uma Operação

54

- Nome da operação
- Visibilidade (por exemplo, “pública”)
- Lista de parâmetros
- Tipo do retorno

```
public int fazAlgo(float nomeDoParametro)
```

# Como nomear uma Operação

55

- ❑ Cada operação deve realizar um serviço
  - ▣ Se você está com dificuldade para nomear uma operação, é provável que exista mais de um serviço oferecido pela operação
  - ▣ Muitos parâmetros também podem indicar múltiplos serviços
  - ▣ Lembre-se da **coesão**

# Antes de continuar...

56

- ❑ Lembre-se que outros programadores precisarão dar manutenção no seu código e na sua documentação
- ❑ O código deve pertencer a todos
  - ▣ Qualquer um deve conseguir entender e manter qualquer parte do código



# Vamos colocar a mão na massa!!!

- ❑ Quais características estão presentes em qualquer pessoa?
- ❑ Considere o contexto de um Sistema Acadêmico



# Uma pessoa pode ter...



Nome

Idade

Data de nascimento

Endereço residencial

Endereço para correspondência

Nome da mãe

CPF

# Você pensou também...

59

- ❑ Sobre o que você precisará fazer com uma pessoa no sistema acadêmico?
- ❑ Se todas as características são necessárias?
  - ❑ Por exemplo, precisamos de um atributo “idade”?
- ❑ Qualquer pessoa tem CPF?
- ❑ ...

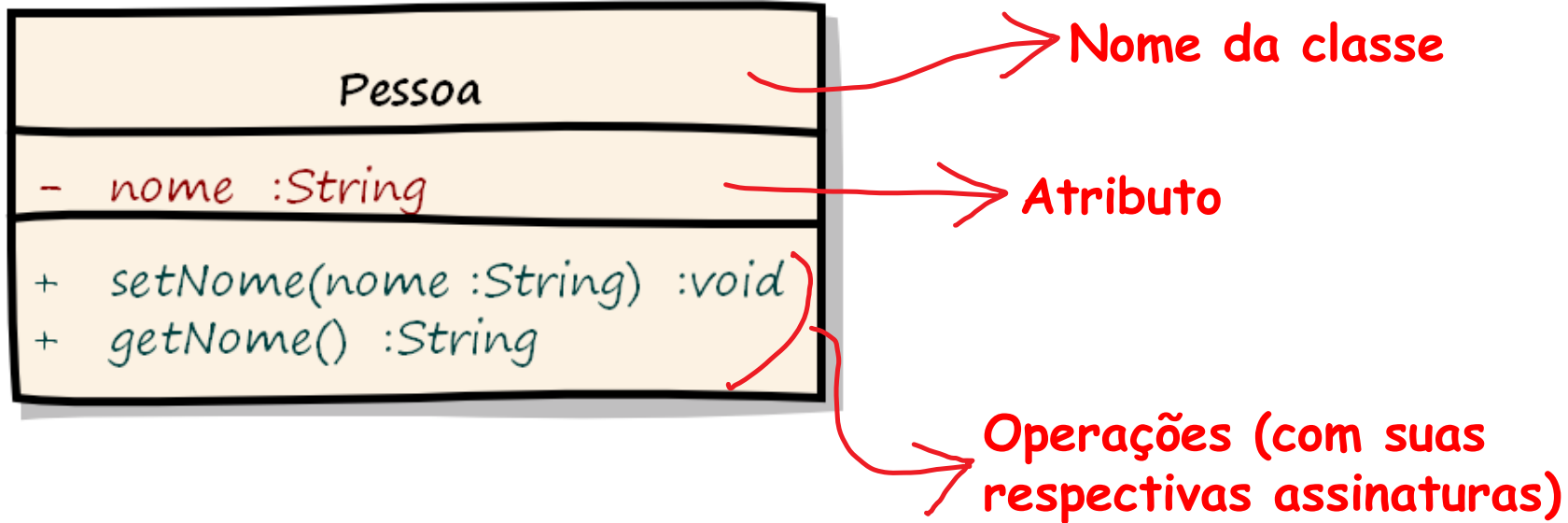
# Por enquanto...

60

- ❑ Vamos considerar que qualquer pessoa precisa:
  - ▣ Ter um nome
  - ▣ Saber informar seu nome
  - ▣ Permitir que seu nome seja alterado
  
- ❑ Vamos abstrair a **classe Pessoa**

# Representação gráfica em UML

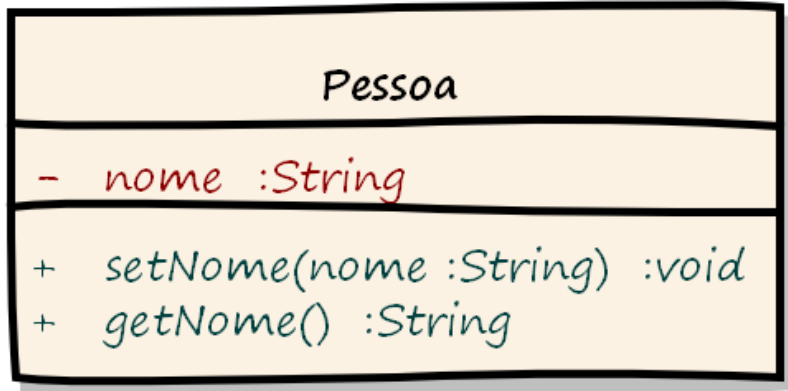
61



□ **UML = Unified Modeling Language**

▣ Linguagem de modelagem → notação visual

# Classe Pessoa: UML para Java



```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

# Classe Pessoa em Java

63

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

→ Nome da classe

→ Atributo

```
    public void setNome(String nome) {  
        this.nome = nome;  
    }
```

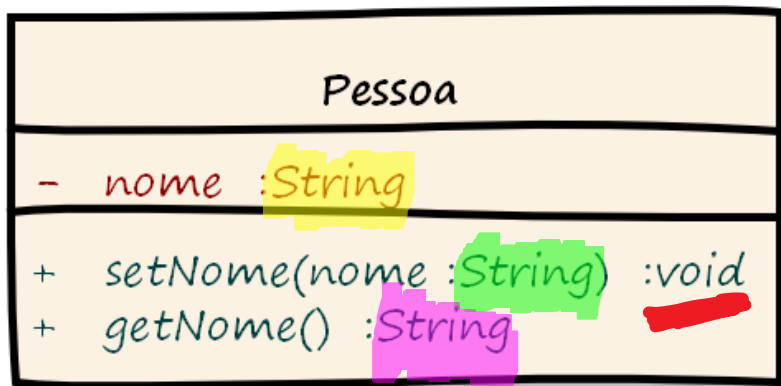
→ Métodos

```
    public String getNome() {  
        return this.nome;  
    }
```

```
}
```

# Traduzindo os tipos

64



A UML permite utilizar os tipos da própria linguagem

```
public class Pessoa {
    private String nome;

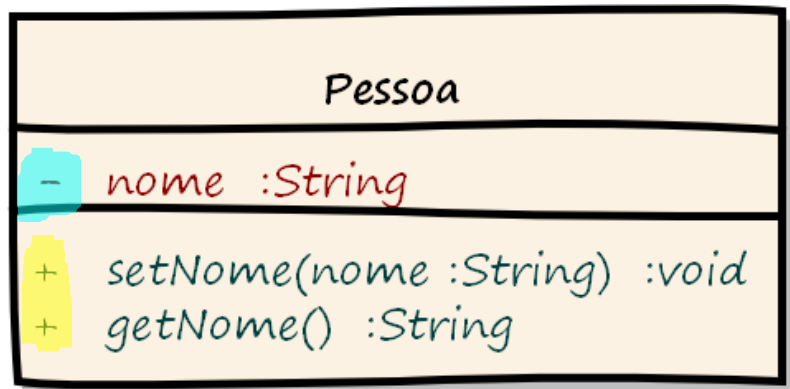
    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return this.nome;
    }
}
```



# Visibilidade

65



A visibilidade indica quem pode ter acesso a um atributo ou operação

```
public class Pessoa {
    private String nome;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return this.nome;
    }
}
```

# E o construtor?

66

```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

Pessoa();



**Default: retorna  
um objeto Pessoa**



# E o destrutor?

67

```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```



**Java usa o conceito de  
Garbage Collector**



# Garbage Collector na linguagem Java

68

- ❑ O próprio ambiente é responsável pelo gerenciamento de memória:
  - ▣ Quando um objeto não é mais usado (ou seja, **não existem mais referências para ele**), ele é finalizado e então pego pelo coletor do lixo (*garbage collector*)
  - ▣ A destruição de um objeto ocorre de **modo assíncrono e em background**

# Mas, vamos manter a calma...

69

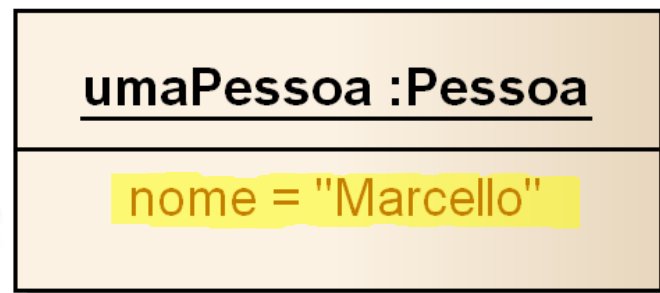
- ❑ Voltaremos a discutir mais sobre construtores e destrutores em Java!!



# Criando objetos a partir de Pessoa

70

```
Pessoa umaPessoa;  
umaPessoa = new Pessoa();  
umaPessoa.setNome("Marcello");
```



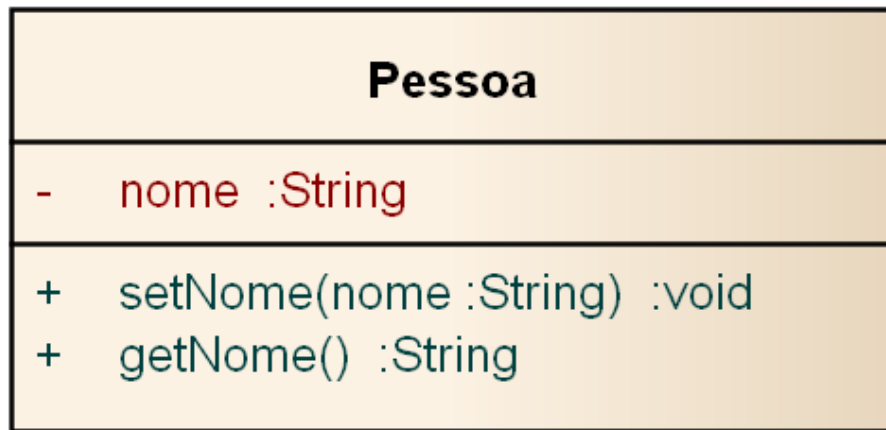
```
Pessoa outraPessoa = new Pessoa();  
outraPessoa.setNome("João");
```

**Estado do  
objeto  
umaPessoa**

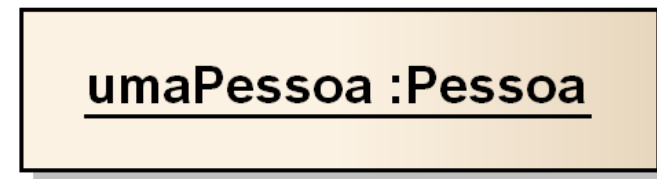
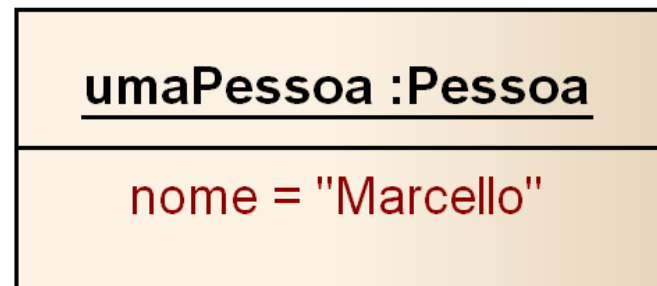
# Classes e Objetos em UML

71

## Classe



## Objeto





# Adicionando comportamento

72

## □ Lembra da idade?

### ▣ Não é um atributo real!

- Embora pareça um atributo real, a idade pode ser calculada
- Idade = atributo virtual (calculado)

### ▣ Seria útil?

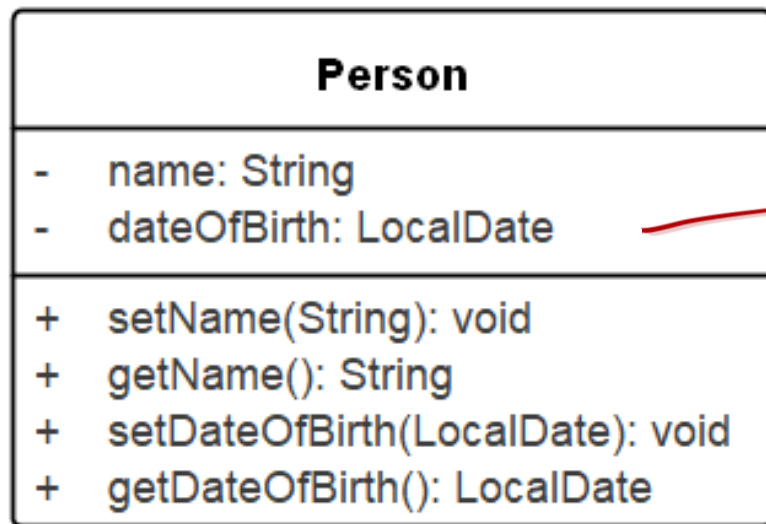
- Você já pensou em como fazer?





# Revisando a modelagem

73

 java.time.LocalDate

# Adicionando o atributo dataNascimento

74

```
private LocalDate dateOfBirth;


public void setDateOfBirth(LocalDate dateOfBirth) {
    this.dateOfBirth = dateOfBirth;
}

public LocalDate getDateOfBirth() {
    return dateOfBirth;
}
```

# Adicionando o atributo dataNascimento

75

```
private LocalDate dateOfBirth;  
  
public void setDateOfBirth(LocalDate dateOfBirth) {  
    this.dateOfBirth = dateOfBirth;  
}  
  
public LocalDate getDateOfBirth() {  
    return dateOfBirth;  
}
```



Você já  
percebeu  
o que a  
palavra  
reservada  
"this"  
faz aqui?



# A palavra reservada “this”

76

- ❑ Dentro de uma operação de objeto ou de um construtor, “**this**<sup>1</sup>” é uma **referência** para o **objeto corrente**
  - ▣ O objeto corrente é aquele para o qual a operação ou construtor está sendo chamada
  - ▣ Você pode se referenciar a qualquer membro (atributo ou operação)

1. A palavra “this” também é adotada em outras linguagens, como C++. Entretanto, a linguagem Object Pascal (ex: para quem utiliza Delphi) adota o termo “self” com o mesmo objetivo.

# Diferenciando atributo e parâmetro...

77

```
public void setNome(String nome) {  
    this.nome = nome;  
}
```

```
public String getNome() {  
    return this.nome;  
}
```

A razão mais comum de utilizar "this" é que um atributo pode estar "escondido" por um parâmetro passado à operação ou construtor

# getIdade() para o atributo simulado

78

```
public int getAge() {  
    Period period = Period.between(dateOfBirth, LocalDate.now());  
    return period.getYears();  
}
```



# Revisando o conceito de construtor

79

- ❑ Construtor é uma operação especial invocada para criar uma nova instância da classe (objeto)
  - ▣ Você pode inicializar os atributos
    - Permite a configuração de um estado inicial para o objeto
  - ▣ Tipicamente, você pode implementar mais de um construtor para a mesma classe
    - Permite diferentes formas de inicializar um novo objeto



# Revisando o conceito de construtor

80

- ❑ Na maioria das linguagens OO, o construtor possui o mesmo nome da classe<sup>1</sup>
- ❑ Por exemplo, para instanciarmos um novo objeto em Java, podemos utilizar o construtor default:

```
Pessoa pessoa = new Pessoa();
```

1. Na linguagem Object Pascal (para quem usa Delphi) o construtor é declarado com a palavra-chave “constructor” e o nome “Create” é comumente adotado, mas é possível utilizar outros nomes.



# Construtor em Java

81

- Já vimos que podemos utilizar o construtor default em Java para criar um novo objeto, utilizando a estrutura:  
`objeto = new NomeClasse();`
- Você pode ainda adicionar comportamento no construtor

```
public Pessoa() {  
    // inicializações desejadas  
}
```

# Construtor em Java

- ❑ Podemos ainda aumentar o poder do construtor, passando parâmetros na criação do objeto:

```
public Pessoa() {  
    // inicializações desejadas  
}  
  
public Pessoa(String nome, String cpf) {  
    this.nome = nome;  
    this.cpf = cpf;  
}
```

# Revisando o conceito de destrutor

83

- ❑ Destrutor é uma operação especial invocada para destruir o objeto corrente
- ▣ Liberação da memória ocupada pelo objeto
- ▣ **Liberação de recursos** usados pelo objeto
  - Ex: fechar uma conexão ou um arquivo



# Destruitor em Java

84

- Nós já vimos que:
  - ▣ Quem gerencia a memória é a máquina virtual Java (JVM – *Java Virtual Machine*)
  - ▣ Quando um objeto não é mais referenciado, ele é coletado (em algum momento<sup>1</sup>) pelo *Garbage Collector*
  - ▣ Desta forma, a linguagem Java não oferece uma operação especial para destruir um objeto explicitamente

1. Não há como saber quando o JVM acionará o Garbage Collector.

# Destruitor em Java

85

- Mas, antes de um objeto ser coletado, a JVM dá uma chance a ele de fazer sua própria limpeza
  - ▣ Este passo é chamado de finalização, onde a JVM invoca a operação “**finalize**” do objeto<sup>1</sup>

```
protected void finalize () {  
    // libere aqui os recursos desejados;  
}
```

1. Mas, não há garantia de que o finalize será sempre chamado!

# Destruitor em Java

86

- ❑ Para forçar a finalização de todos os objetos que estão aguardando pelo *Garbage Collector*, você pode utilizar:

```
System.runFinalization();
```

- ❑ Você pode ainda forçar a execução do *Garbage Collector*:

```
System.gc();
```

*Nota: Devemos evitar chamadas diretas a estas operações; veja também se você realmente precisa do finalize.*

# Destruitor em outras linguagens OO

87

- ❑ Em **C++**, a convenção é utilizar o mesmo nome da classe com o prefixo til (ex: ~Pessoa)
- ❑ Em **Object Pascal**, um destrutor é declarado com a palavra chave "destructor" e o nome "Destroy" é comumente adotado, mas é possível utilizar outros nomes
- ❑ Em **Perl**, o destrutor é nomeado "DESTROY"
- ❑ A partir do **PHP5**, o destrutor é nomeado "\_\_destruct"

# Revisando o conceito de atributo


88

- ❑ Antes de mais nada, você saberia explicar o que é um atributo?
  - ▣ Somente atributos que são de interesse do sistema devem ser descritos na classe (pense na relevância do atributo)
- ❑ **Dica:** um atributo deve ser declarado somente quando a necessidade da informação armazenada por este atributo for maior do que a duração de uma operação



# Revisando o conceito de atributo

89



Na classe Pessoa, o atributo "nome" informado pela operação "setNome" precisa ser armazenado, pois poderá ser utilizado em outro momento pela operação "getNome"

- ❑ **Dica:** um atributo deve ser declarado somente quando a necessidade da informação armazenada por este atributo for maior do que a duração de uma operação



# Revisando o conceito de atributo

90

- ❑ Até o momento, nós trabalhamos basicamente com **atributos de objeto**
  - ▣ Definem informações específicas a cada objeto
  - ▣ Cada objeto possui sua própria cópia (com um respectivo valor) do atributo
  - ▣ Juntamente com seus valores, estas informações estabelecem o estado do objeto



# Atributo de objeto

91

- Na classe “Pessoa”, o atributo “nome” é um atributo de objeto
  - ▣ Cada objeto “Pessoa” possui sua própria cópia do atributo
  - ▣ Cada objeto “Pessoa” tem um valor próprio para o atributo “nome”

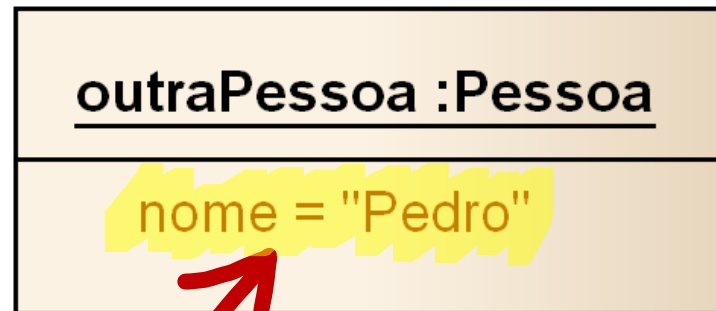
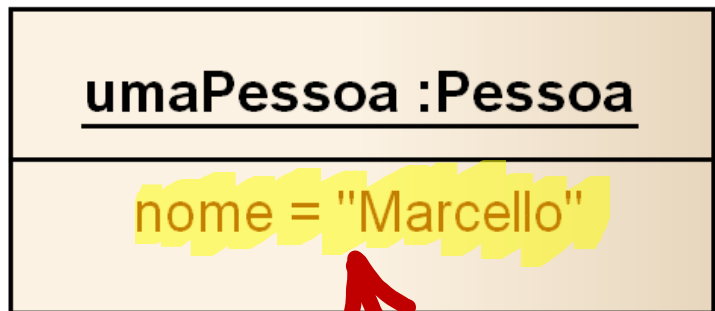
Pessoa	
-	nome :String
+	setNome(nome :String) :void
+	getNome() :String

<u>umaPessoa :Pessoa</u>
nome = "Marcello"

<u>outraPessoa :Pessoa</u>
nome = "Pedro"

# Atributo de objeto

92



Alterar o valor do atributo "nome" no objeto "umaPessoa" não afetará o atributo "outraPessoa" (cada objeto é uma pessoa diferente com seu próprio nome)



# Um novo tipo de atributo...



- Um **atributo de classe** é uma característica cujo valor é compartilhado por todos os objetos de uma mesma classe
  - ▣ Também conhecido como atributo estático



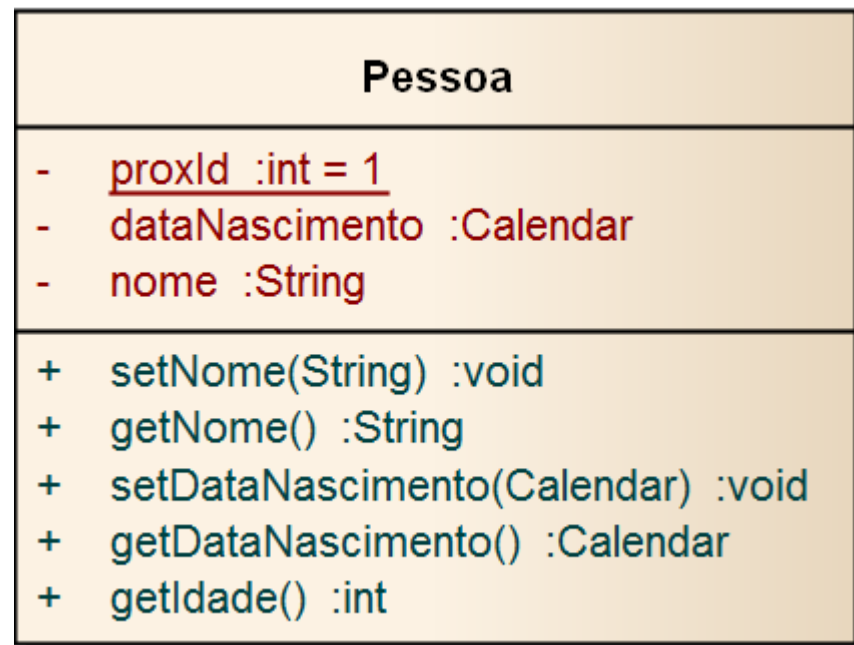
# Atributo de classe

94

- ❑ Considerado estático, pois não há alocação dinâmica de memória
  - ▣ Ao declarar a classe, o atributo já passa a ocupar memória e já pode ser acessado
    - O atributo pode ser acessado diretamente a partir da classe, sem a necessidade de instanciar previamente um objeto
  - ▣ Todos os objetos instanciados acessam o mesmo atributo/valor

# Atributo de classe na UML

- Na UML, um atributo de classe (atributo estático) aparece sublinhado
- ▣ O valor de “**proxId**” sempre será o mesmo para todos os objetos “Pessoa” em um determinado momento no tempo



# Atributo de classe em Java

96

```
public class Pessoa {  
  
    private static int proxId = 1;  
    private int id;  
    private String nome;  
    private Calendar dataNascimento;  
  
    public Pessoa() {  
        id = proxId++;  
    }  
  
    ...  
}
```

Modificador usado  
no atributo para  
indicar que ele é  
um **Atributo de  
Classe** (Atributo  
estático)



# Atributo de classe em Java


97

```
public class Pessoa {  
  
    private static int proxId = 1;  
    private int id;  
    private String nome;  
    private Calendar dataNascimento;
```

```
    public Pessoa() {  
        id = proxId++;  
    }
```

...

O que  
acontece  
aqui?



# Atributo de classe em Java

98

Considerando que foram criados 3 objetos Pessoa...

```
Pessoa pessoa1 = new Pessoa();
```

```
Pessoa pessoa2 = new Pessoa();
```

```
Pessoa pessoa3 = new Pessoa();
```

*Nota: Os nomes adotado para os objetos tem caráter apenas didático, para facilitar as referências.*

# Atributo de classe em Java



```
Pessoa pessoa1 = new Pessoa();
```

pessoa1 :Pessoa

proxId = 2

id = 1

# Atributo de classe em Java



100

```
Pessoa pessoa2 = new Pessoa();
```

pessoa1 :Pessoa

proxId = 3  
id = 1

pessoa2 :Pessoa

proxId = 3  
id = 2

# Atributo de classe em Java

101

```
Pessoa pessoa3 = new Pessoa();
```

pessoa1 :Pessoa

proxId = 4  
id = 1

pessoa2 :Pessoa

proxId = 4  
id = 2

pessoa3 :Pessoa

proxId = 4  
id = 3

Pegou a ideia?

# Atributo de classe em Java

102

```
public Pessoa() {  
    this.id = this.proxId++;  
}
```



Ok?

# Atributo de classe em Java

103

```
public Pessoa() {  
    this.id = this.proxId++;  
}
```



Ok?

Mas, não é uma boa prática  
para atributos de classe

# Revisando o conceito de operação

104

- ❑ Construtoras
  - ▣ Instanciar/criar objetos
- ❑ Destrutoras
  - ▣ Liberar/destruir objetos (em Java, temos o *Garbage Collector*)
- ❑ Modificadoras
  - ▣ Modificar o estado (parcial ou total) de objetos (ex: setter's)
- ❑ Recuperadoras
  - ▣ Recuperar o estado (parcial ou total) de objetos (ex: getter's)





# Revisando o conceito de operação

105

- Até o momento, nós trabalhamos basicamente com **operações de objeto**
  - ▣ Podem manipular atributos de objeto e de classe
  - ▣ O objeto precisa ter sido instanciado para que a operação seja invocada

# Um novo tipo de operação...

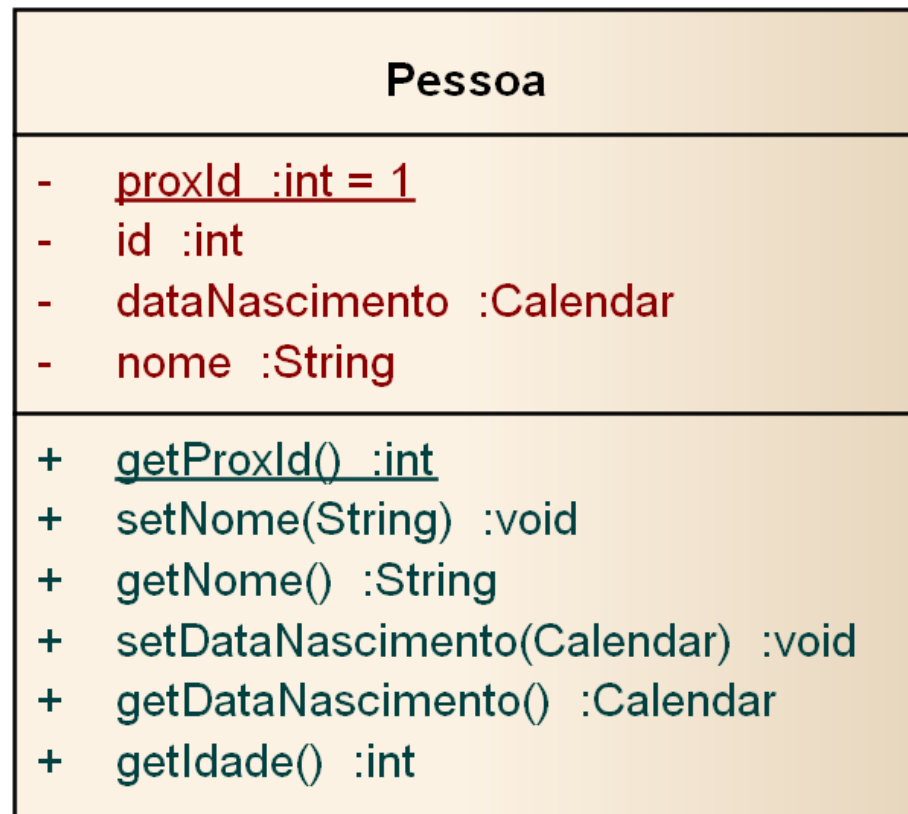
106

- ❑ Uma **operação de classe** é um serviço que pode ser invocado diretamente a partir da classe
  - ▣ Também conhecida como operação estática
  - ▣ Podem manipular somente atributos de classe
  - ▣ Pode ser invocada diretamente pela classe, sem que objetos tenham sido instanciados

# Operação de classe na UML

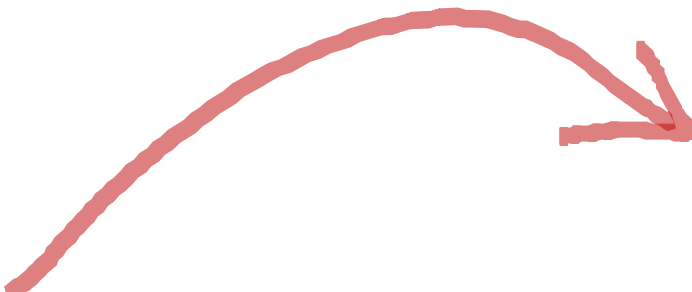
107

- Na UML, uma operação de classe (operação estática) aparece sublinhada
  - ▣ Note a inclusão da operação “**getProxId**”



# Operação de classe em Java

108



```
public static int getProxId() {  
    return proxId;  
}
```

Modificador usado  
na operação para  
indicar que ela é  
uma **Operação de  
Classe** (Operação  
estática)

# Operação de classe em Java

109

```
public static int getProxId() {  
    return this.proxId;  
}
```



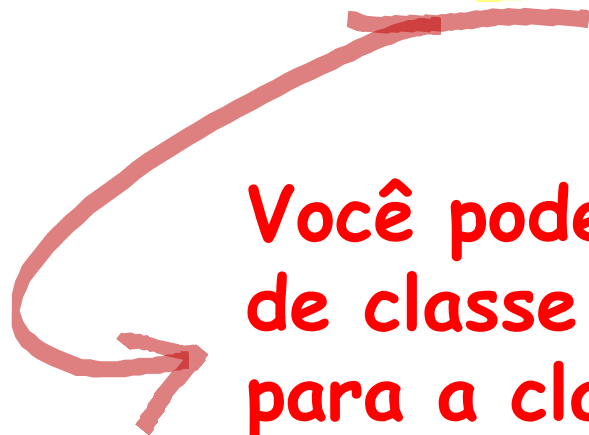
Ok?

Operações de classe não podem manipular atributos de objeto: "this" aponta para o objeto corrente

# Invocando operações de classe em Java

110

```
System.out.println(Pessoa.getProxId());
```



Você pode invocar uma operação de classe pedindo diretamente para a classe

Não é necessário instanciar um objeto antes

# Referências

111

- ❑ Grady Booch; Robert A. Maksimchuck; Michael W. Engle; Bobbi J. Young; Jim Conallen; Kelli A. Houston. **Object-oriented analysis and design with applications**. 3rd ed. Addison-Wesley, 2007.
- ❑ Ricardo Pereira e Silva. **UML 2 em Modelagem Orientada a Objetos**. Visual Books, 2007.
- ❑ Paul J. Deitel; Harvey M. Deitel. **Java - Como Programar**. 8ª ed., Prentice Hall, 2010.
- ❑ OMG (Object Management Group), **OMG Unified Modeling Language v2.5**, 2012.
  - ❑ <http://www.omg.org/spec/UML/2.5/Beta1/PDF/>

Clássico!!  
Excelente  
referência!

Padrão  
UML!