

Programação Orientada a Objetos

Classes e Objetos

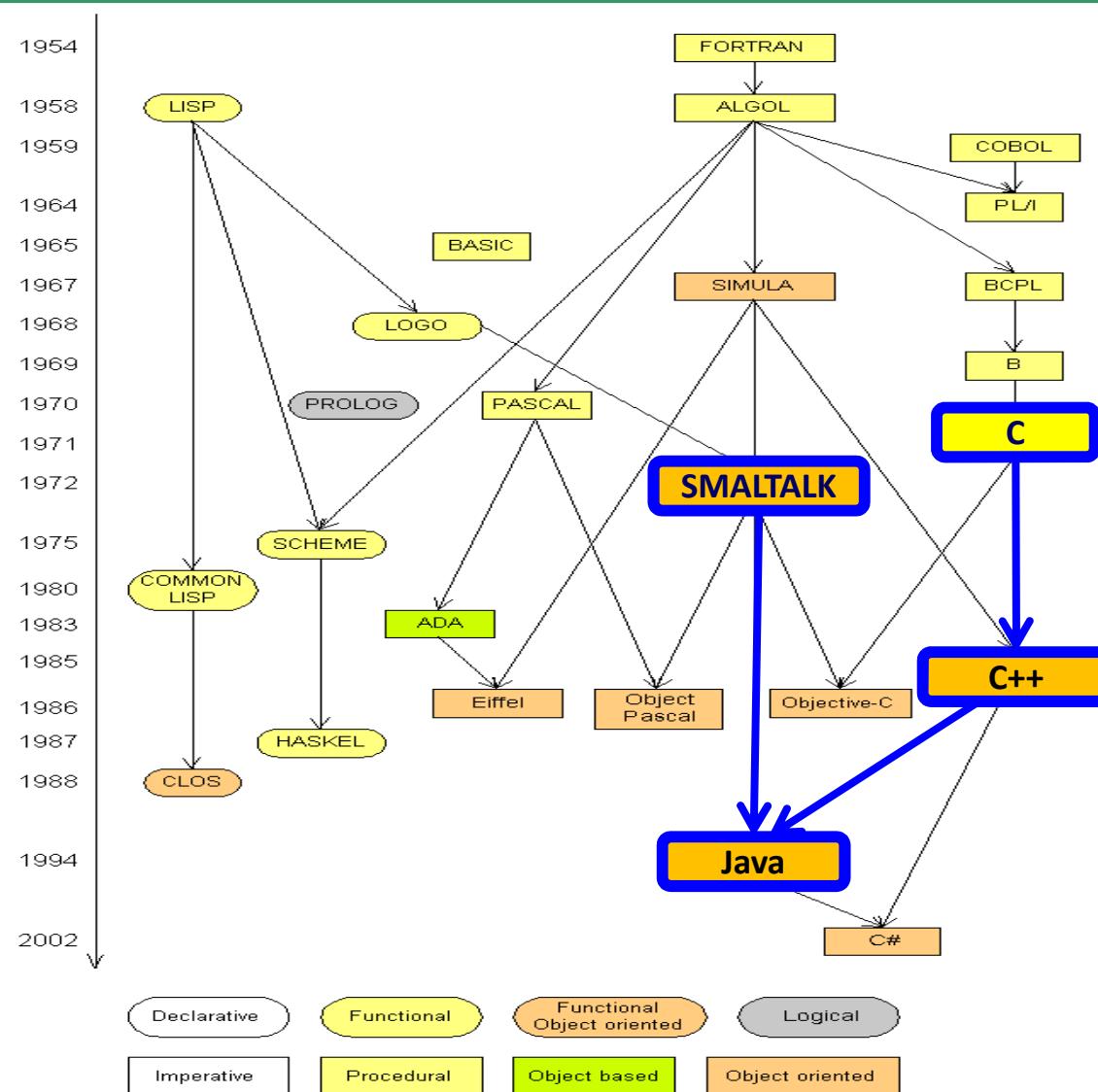
Prof. Rui César das Neves, Prof. José Cordeiro

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal
2014/2015

Sumário

- História e evolução das linguagens de programação
- O paradigma orientado por objetos
- Classes e objetos
 - Estado (Atributos)
 - Comportamento (Métodos)
 - Troca de Mensagens
 - Estado privado versus Comportamento público
 - JAVA - Métodos básicos
 - construtores
 - inspetores ou seletores
 - modificadores

Evolução das Linguagens de Programação



Problemas das Técnicas Procedimentais

- Não lidam adequadamente com o problema da complexidade e do tamanho crescente dos sistemas.
- Não resolvem o problema da crescente atividade de manutenção de software.
 - Nomeadamente não lidam adequadamente com as alterações de requisitos.
 - Os erros de concepção são descobertos tarde e a sua correção implica normalmente rever a globalidade do código.
 - Não é fácil identificar quem faz o quê, quando e porquê (legibilidade do código).
- A integração e reutilização de módulos e componentes do sistema não é fácil.

Evolução das Linguagens de Programação

Resumindo:

Linguagens	Base	Programação
FORTRAN, BASIC	Instrução	Não estruturada
PASCAL, C	Procedimentos	Estruturada
MODULA	Modulo	Modular
ADA	Modulo + “ADT”	Modular com “ADT”
SMALTALK, C++; Java; C#	Objetos Classes/Objetos	Por objetos

- Ao longo da segunda metade do século XX a evolução dos paradigmas de programação tem tido como principais motores evolutivos:
 - A crescente necessidade de produzir código reutilizável independentemente do contexto.
 - A crescente necessidade de produzir código fácil de manter (resistente à mudança, que simplifique a procura e correção de erros e facilite a evolução modular dos sistemas)
 - O objetivo é atingido em grande parte com o paradigma da **Programação Orientada a Objetos** no fim do Século XX

Paradigma Orientado por Objetos

□ Característica Distintiva:

- **Encapsulamento do Comportamento** – Os objetos devem ser considerados metáforas de comportamento de entidades reais
 - (ex: Relógio, Telemóvel, Pessoa, ...)

□ Motivação:

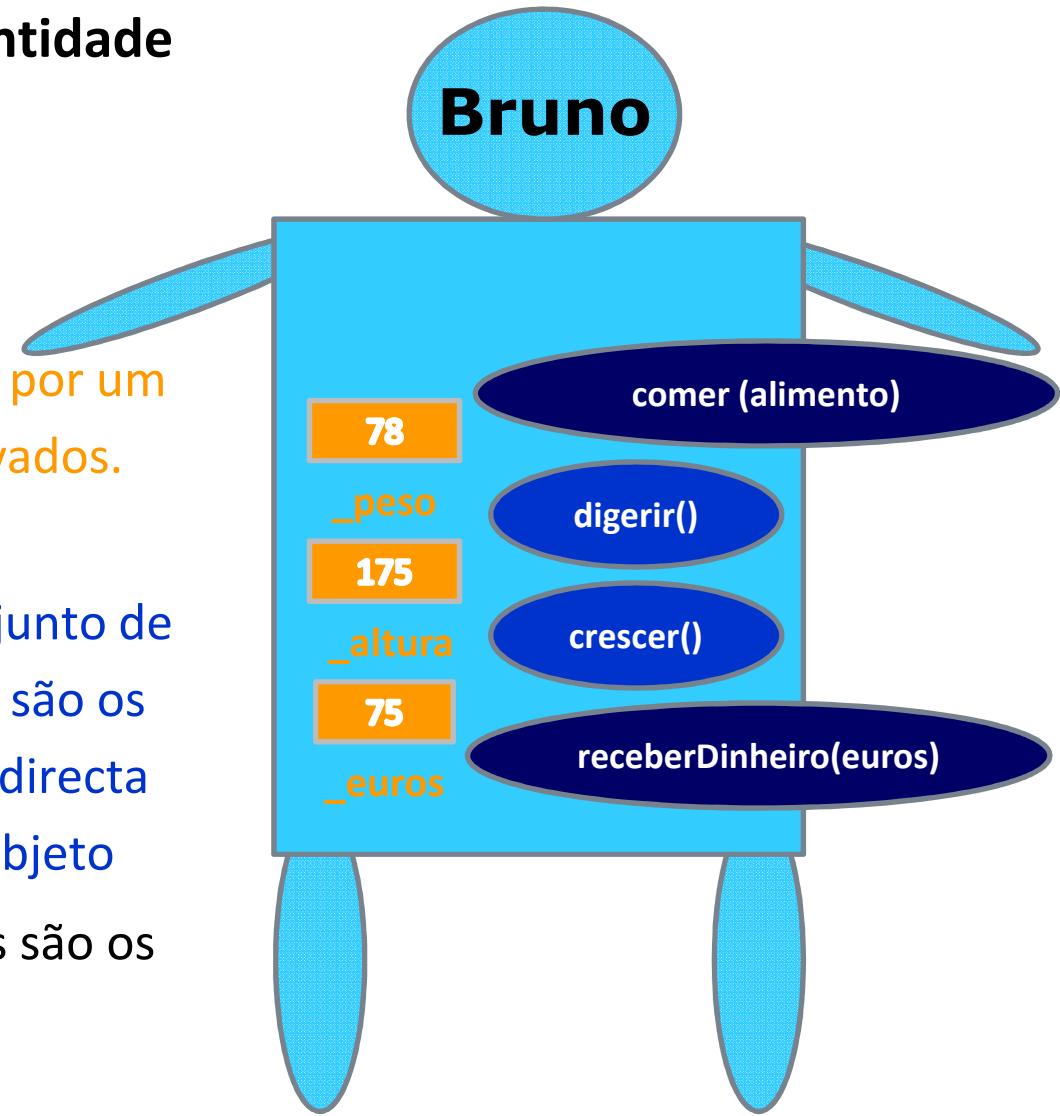
- Poder **tratar** domínios de **problemas** mais **complexos**;
- Melhorar a **interacção** entre o **programador/analista** e **cliente/especialista** do problema;
- Representar **explicitamente** aspectos comuns a diversos conceitos;
- Construir especificações capazes de **resistir à mudança**;

Objeto (para quê?)

- O conceito de **objeto** pretende responder às exigências de um modelo de concepção e desenvolvimento de software que resolva os problemas anteriormente identificados e para isso deve garantir:
 - A **Abstracção de Dados** (escondendo a sua estrutura)
 - A **Modularidade** (permitindo a composição de partes)
 - A **Independência de Contexto** (Objetivo dos dois anteriores)
- O **Encapsulamento** (solução oferecida pelo paradigma OO)

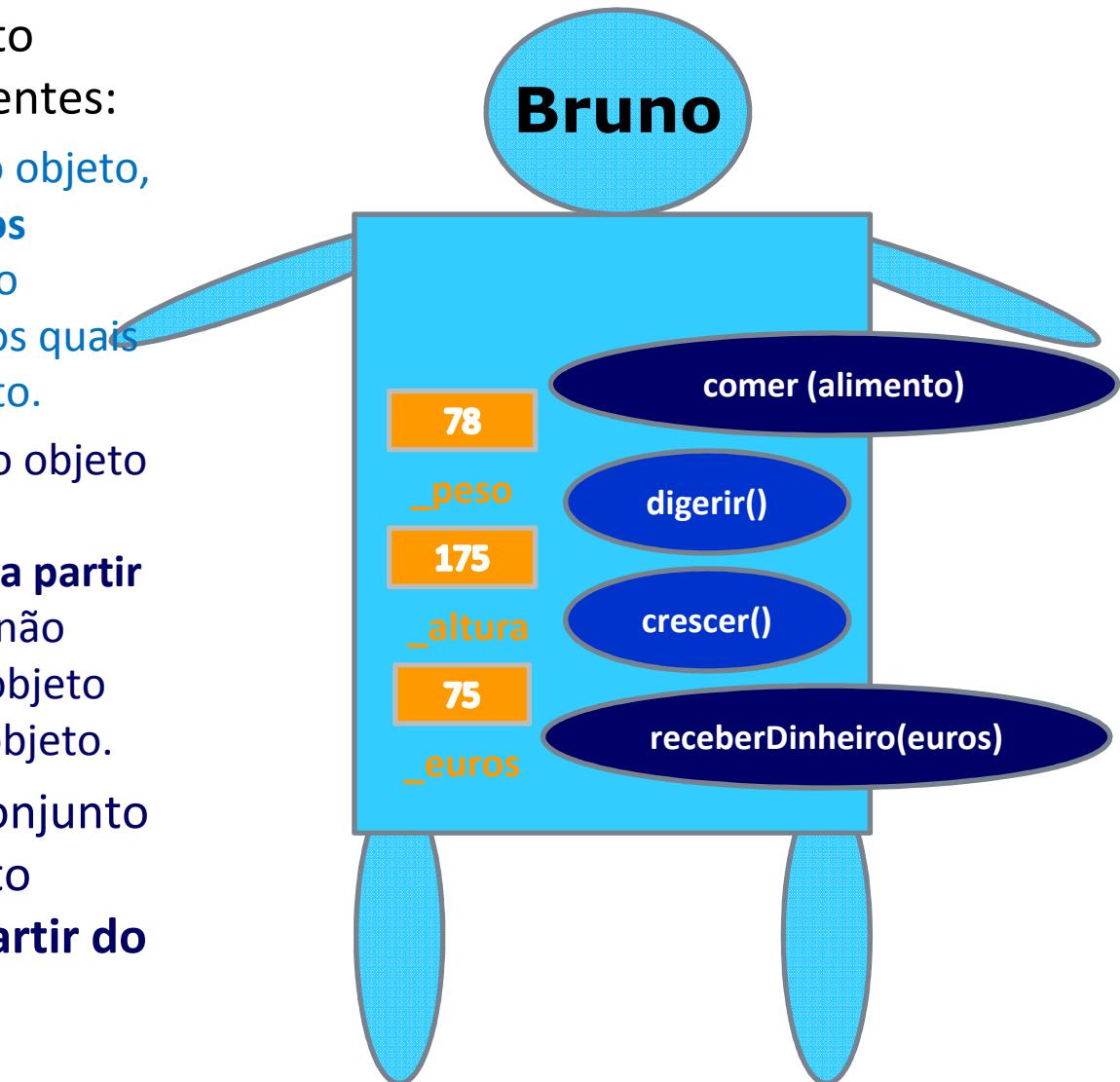
Objeto (O que é?)

- É uma abstração de uma entidade real e autónoma.
- É uma entidade única.
- É caracterizada por:
 - Um Estado: representado por um conjunto de **atributos** privados.
 - Um Comportamento: representado por um conjunto de **métodos** (operações) que são os únicos a aceder de forma directa ao estado (**atributos**) do objeto
 - Os atributos e os métodos são os **membros** de um objeto.



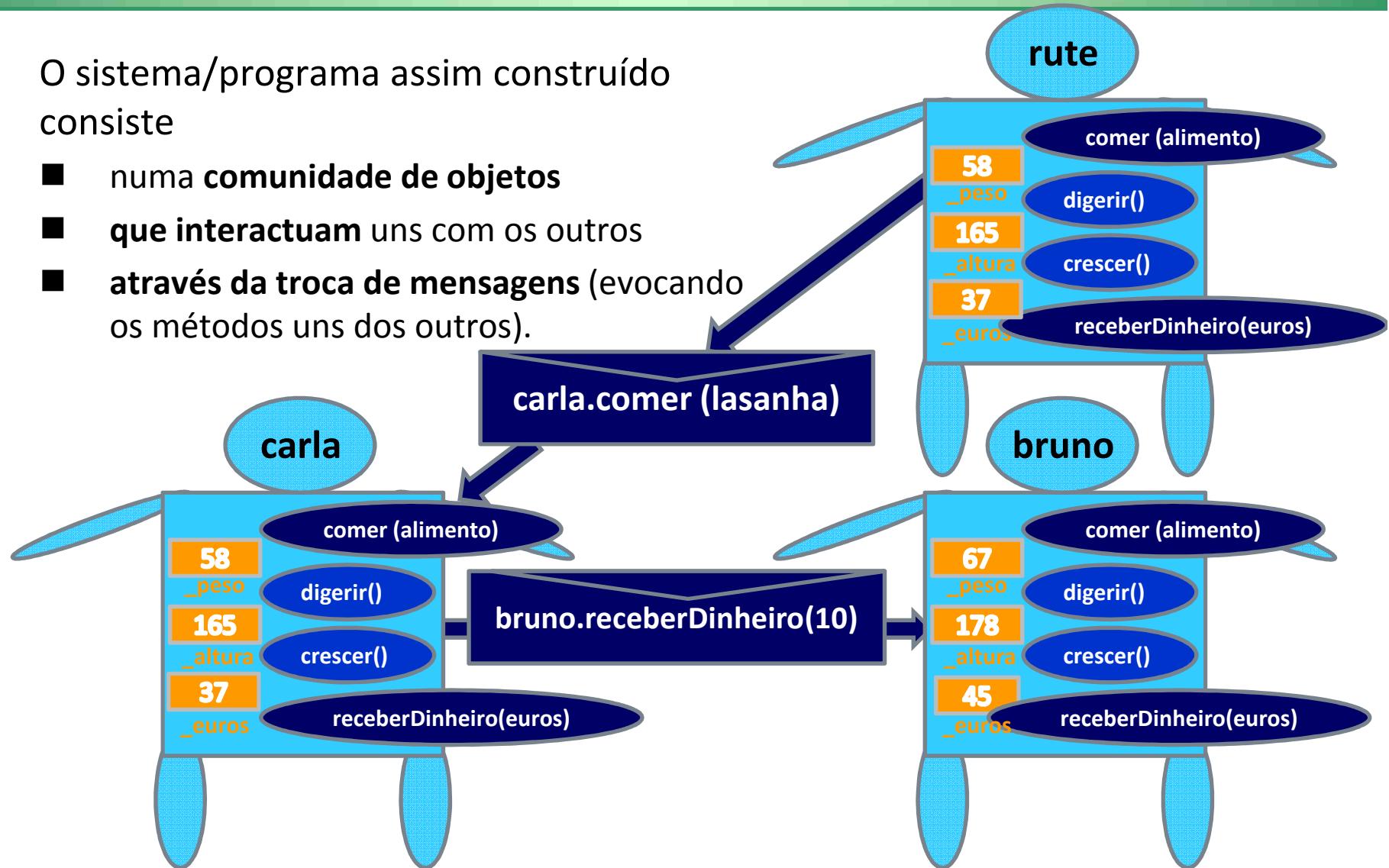
Objeto (Tem comportamento!)

- O Comportamento do objeto divide-se em duas componentes:
 - **O comportamento interno** do objeto, ou seja o conjunto de **métodos internos, privados**, aos quais o exterior não pode aceder e dos quais nem sequer tem conhecimento.
 - **O comportamento externo** do objeto constituído pelo conjunto de **métodos públicos, acessíveis a partir do exterior**, e que podem ou não aceder ao estado interno do objeto ou aos métodos internos do objeto.
- A **interface** do objeto é o conjunto de **operações** que um objeto define como **acessíveis a partir do exterior**.



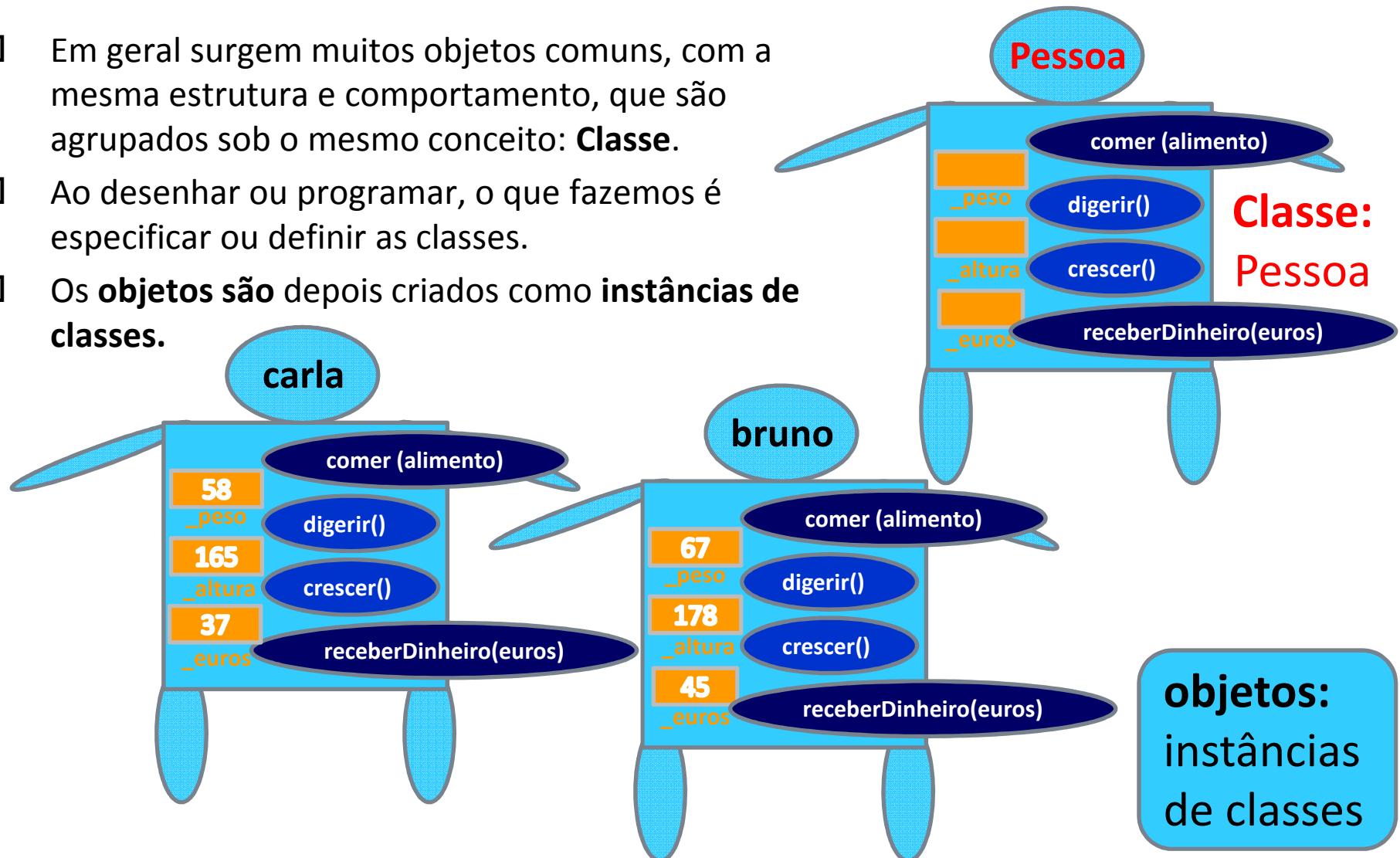
Objetos (interatuam)

- O sistema/programa assim construído consiste
 - numa **comunidade de objetos**
 - que **interactuam** uns com os outros
 - através da **troca de mensagens** (evocando os métodos uns dos outros).



Classe

- Em geral surgem muitos objetos comuns, com a mesma estrutura e comportamento, que são agrupados sob o mesmo conceito: **Classe**.
- Ao desenhar ou programar, o que fazemos é especificar ou definir as classes.
- Os **objetos** são depois criados como **instâncias** de classes.



Programação Orientada por Objetos ...

- Uma forma (mais realista?) de ver o mundo.

- Implica que organizemos a nossa visão do **mundo** e/ou de um problema **em termos dos objetos envolvidos**, dos seus atributos e operações, tanto internas ao objeto como acessíveis a partir do seu exterior.

- Implica que olhemos para a **resolução de um problema** como a **procura dos objetos envolvidos e da forma de os por a cooperar** (trocando mensagens) por forma a resolver o problema.

Classes e Objetos

Em Resumo:

- **Uma Classe** representa todos os potenciais objetos que partilham um conjunto de atributos e métodos (define um novo tipo na linguagem)
- **Um objeto** representa uma entidade no mundo real que pode ser distintamente identificada:
 - Identificador único (a sua referência)
 - Estado interno
 - Um comportamento específico que pode depender do estado interno.
- **Um programa** é constituído por conjuntos de objetos que comunicam entre si, trocando mensagens, por forma a resolver um problema.

Criação de objetos

□ Operador **new**

- Criação de objetos em 3 passos:

□ 1. Definição da referência:

- A) Cria na memória a variável bruno
- B) Especifica que essa variável refere um objeto do tipo Pessoa.
- C) Coloca “null” na variável

□ 2. Instanciação:

- D) Reserva espaço em memória para o objeto.
- E) O objeto é criado nesse espaço.

□ 3. Referenciação:

- F) Escreve na referência o endereço de memória onde começa o objeto

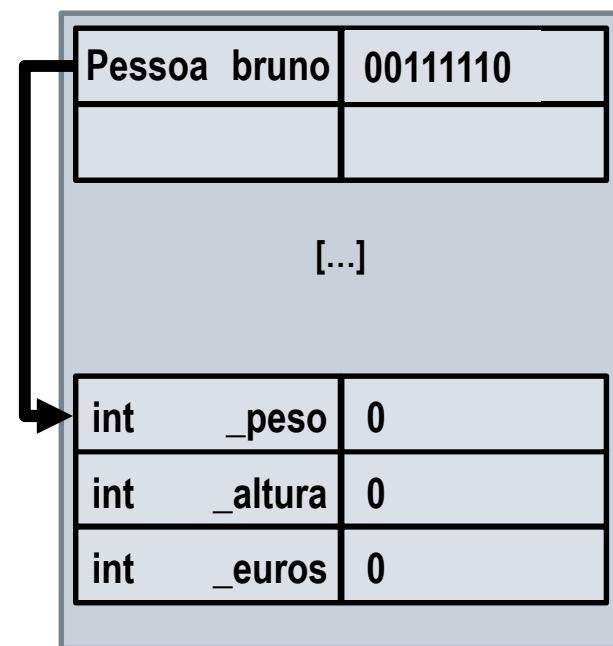
□ Código:

```
//Definição  
Pessoa bruno;
```

```
//Instanciação  
bruno = new Pessoa();
```

```
//Referenciação  
bruno = new Pessoa();
```

□ Ilustração da Memória:



Criação de objetos

□ Operador **new**

■ **Uma Linha os mesmos 3 passos:**

□ 1. Definição da referência:

- A) Cria na memória a variável bruno
- B) Especifica que essa variável refere um objeto do tipo Pessoa.
- C) Coloca “null” na variável

□ 2. Instanciação:

- D) Reserva espaço em memória para o objeto.
- E) O objeto é criado nesse espaço.

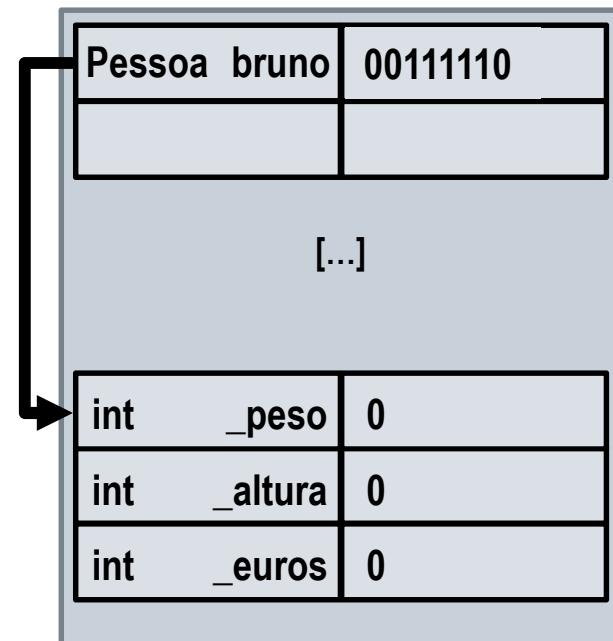
□ 3. Referenciação:

- F) Escreve na referência o endereço de memória onde começa o objeto

□ Código:

//Definição+Instanciação+Referenciação

Pessoa bruno = new Pessoa();



3 Tipos de Métodos básicos

- **Construtores:** métodos que criam os objetos por instanciação das classes, inicializando os seus atributos.
 - Os construtores têm por identificador o nome da classe, podendo ou não aceitar parâmetros na sua lista de parâmetros.
 - Ex: **Pessoa()** ou **Pessoa(int altura, int peso)**
- **Inspetores (ou seletores):** são os métodos que permitem obter o estado do objeto e que não o alteram (devolvem o valor de um ou mais atributos).
 - Por convenção, os inspectores têm, no seu nome, o prefixo **get**. (Exceção do **is**!)
 - Ex: **int getAltura();** Ex: **boolean isObeso();**
- **Modificadores:** métodos que alteram o estado do objeto (valor dos atributos).
 - Por convenção, os métodos que alteram explicitamente o valor de um atributo têm, no seu nome, o prefixo **set**.
 - Ex: **void setAltura(int altura)**

Exemplo Java

Java - OO

```
public class Pessoa {  
    private int _peso;  
    private int _altura;  
    // Construtores com e sem parâmetros  
    public Pessoa() { _peso = 0; _altura = 0 }  
    public Pessoa (int peso, int altura) {  
        _peso = peso;  
        _altura = altura;  
    }  
    // Inspectores ou Seletores  
    public int getPeso () { return _peso; }  
    public int getAltura () { return _altura; }  
    // Modificadores  
    public void setPeso (int peso) { _peso = peso; }  
    public void setAltura (int altura) { _altura= altura; }  
}
```

Acesso aos membros dos objetos

- Operador • (ponto)
`identificador.atributo` ou `identificador.metodo()`
- Só se pode aceder a um membro de um objeto depois do objeto ter sido instanciado e referenciado (i.e., quando na referência do objeto o valor “null” já foi substituído pelo valor do endereço de memória do objeto).

```
Pessoa bruno; // null
bruno = new Pessoa();
bruno.setPeso(72);
Pessoa carla; // null!
carla.setPeso(72); // Erro! Referência é null
```

Exemplo Java

Java - OO

```
public class Pessoa {  
    private int _altura;  
    private int _peso;  
    public Pessoa() { _altura = 0; _peso = 0; }  
    public Pessoa (int peso, int altura) {  
        _altura = altura;  
        _peso = peso;  
    }  
    public int getAltura () { return _altura; }  
    public int getPeso () { return _peso; }  
    public void setAltura (int altura) { _altura = altura; }  
    public void setPeso ( int peso ) { _peso = peso; }  
    public void comer (int quantidadeDeComida) {  
        setPeso( _peso + quantidadeDeComida / 1000 );  
        setAltura( _altura + quantidadeDeComida / 10000 );  
    }  
}
```

Exemplo Java

Java - OO

```
public class Programa {  
  
    public static void main ( String[] args )  
    {  
        Pessoa bruno = new Pessoa();  
        bruno.setPeso ( 71 );  
        bruno.setAltura( 175 );  
  
        System.out.printf("Peso %d e Altura %d\n",  
                           bruno.getPeso(),  
                           bruno.getAltura());  
  
        bruno.comer( 100 );  
  
        System.out.printf("Altura %d e Peso %d\n",  
                           bruno.getAltura(),  
                           bruno.getPeso());  
    }  
}
```

Classe é um módulo

□ Interface:

- Parte pública (declarações públicas, métodos).

□ Implementação:

- Parte privada (declarações privadas, variáveis e métodos).
- Corpo dos métodos (implementação das operações).

□ Manual de utilização

- Comentários de documentação da classe
- Manual de utilização de cada operação pública

Resumindo

□ O Paradigma Orientado por Objetos:

- Encapsulamento do estado e do comportamento interno asseguram a independência do contexto.

□ Classes e Objetos (instâncias das classes).

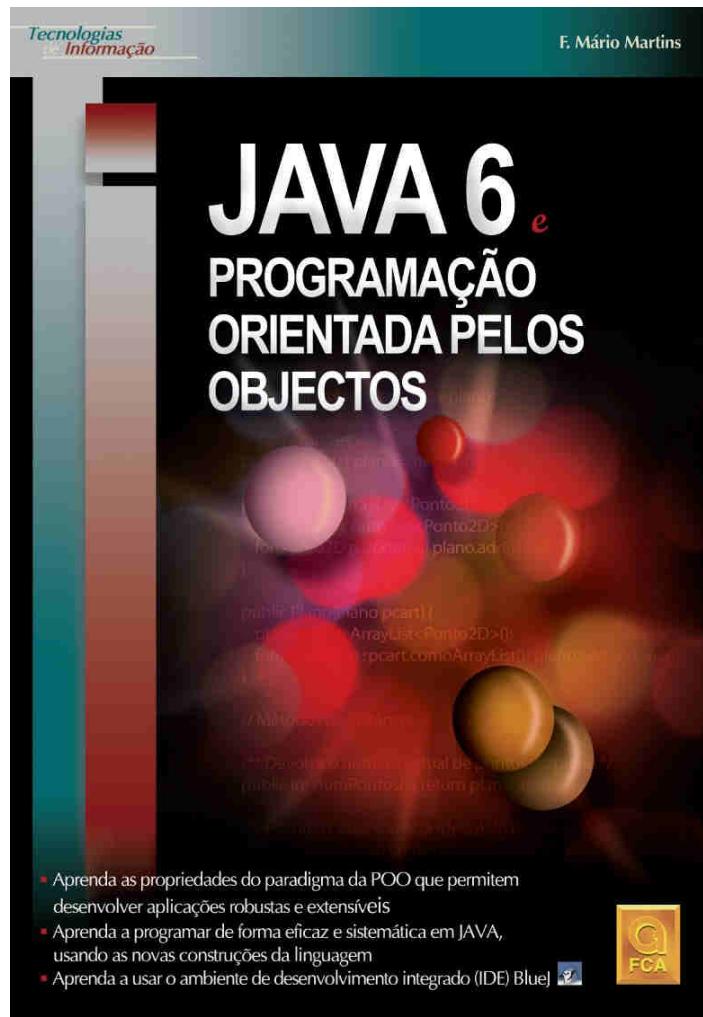
- Estado – definido pelo valor dos atributos (variáveis)
- Comportamento – definido pelos métodos (procedimentos e funções)
- Troca de Mensagens – é como comunicamos! É como os objetos comunicam
- Estado **Privado** versus Comportamento **Público**

□ Java - Métodos básicos

- Construtores – **NomeDaClasse()** { [...] }
- Inspectores – tipo **getAtributo()** { [...] }
- Modificadores – void **setAtributo(tipo atributo)** { [...] }

Leitura Complementar

- Capítulo 1
 - Páginas 1 a 26



Exemplo Java (Relógio)

Java - OO

```
public class Relogio {  
    private int _horas;  
    private int _minutos;  
    private int _segundos;  
  
    public Relogio() { _horas = 0; _minutos = 0; _segundos = 0; }  
    public Relogio (int horas, int minutos, int segundos) {  
        _horas = horas;  
        _minutos = minutos;  
        _segundos = segundos;}  
  
    public int getHoras () { return _horas; }  
    public int getMinutos () { return _minutos; }  
    public int getSegundos () { return _segundos; }  
  
    public void setHoras (int horas) { _horas = horas; }  
    public void setMinutos (int minutos) { _minutos = minutos; }  
    public void setSegundos (int segundos) { _segundos = segundos; }  
  
    public void avancarMinutos (int minutos) {  
        _horas += (_minutos + minutos) / 60;  
        _horas = _horas % 24;  
        _minutos = (_minutos + minutos) % 60; }  
}
```

Exemplo Java (Relógio)

Java - OO

```
public class Programa {  
  
    public static void main ( String[] args )  
    {  
        Relogio timex = new Relogio(10,25,15);  
  
        System.out.printf("Horas %d Minutos %d\n",  
                          timex.getHoras(),  
                          timex.getMinutos());  
  
        timex.avancarMinutos(55);  
  
        System.out.printf("Horas %d e Minutos %d\n",  
                          timex.getHoras(),  
                          timex.getMinutos());  
    }  
  
}
```