Tema 2. POO com UML Java. Desenho de classes.

- Uso da linguagem UML
- Modelos, diagramas, visões e elementos
- Desenho de classes

POO com UML Java

Uso da linguagem UML(*Unified Modeling Language*)

A UML, ou Linguagem de Modelagem Unificada, é a junção das três mais conceituadas linguagens de modelagem orientados a objectos (*Booch de Grady*, OOSE de Jacobson e o OMT de Rumbaugh).

UML não é uma LP. Serve para visualização, especificação, construção e documentação de sistemas que é padrão para modelagem orientada a objetos.

Vantagens da UML:

- É um padrão aberto;
- Suporta todo o **ciclo de vida** do *software*:
  - modelagem do negócio (processos e objectos do negócio)
  - modelagem de requisitos alocados ao *software*
  - modelagem da solução de *software*
- Suporta diversas áreas de aplicação;
- É baseada na experiência e necessidades da comunidade de usuários;
- É suportada por muitas ferramentas.

Modelos, diagramas, visões e elementos

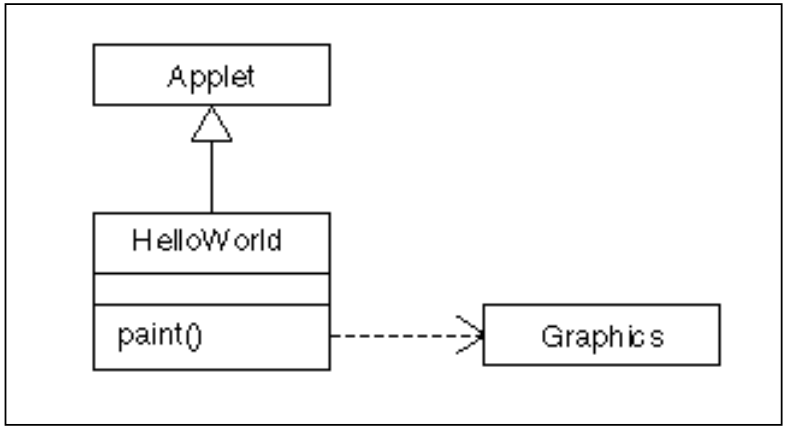
- Um modelo é uma representação em **pequena escala,** numa perspectiva particular, de um sistema existente ou a criar;

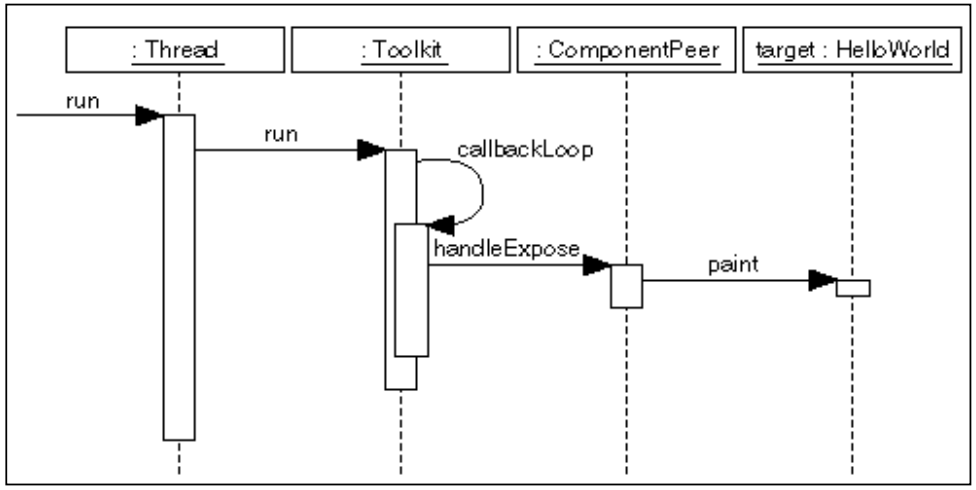
- Ao longo do ciclo de vida de um sistema são construídos vários modelos, sucessivamente refinados e enriquecidos;

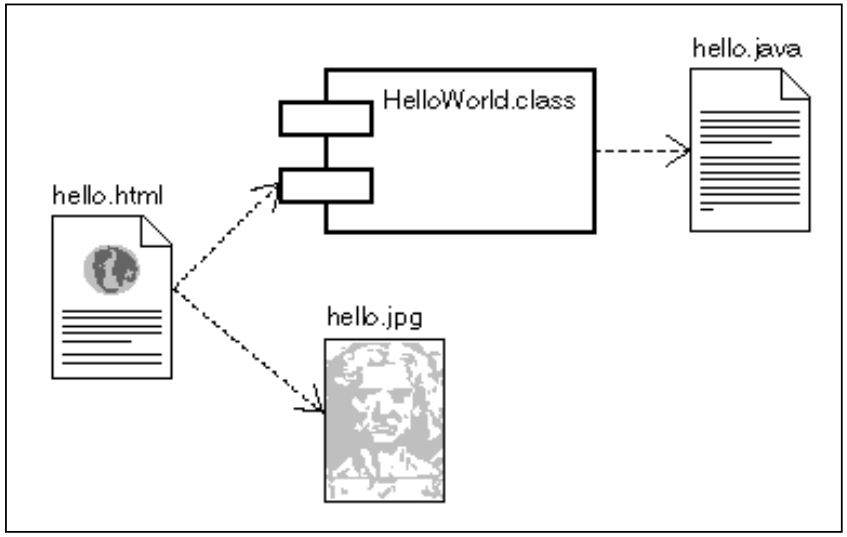
- Um modelo é constituído por um conjunto de **diagramas** (desenhos) consistentes entre si, acompanhados de descrições textuais dos **elementos** que aparecem nos vários diagramas.

Exemplo básico – Hello World!

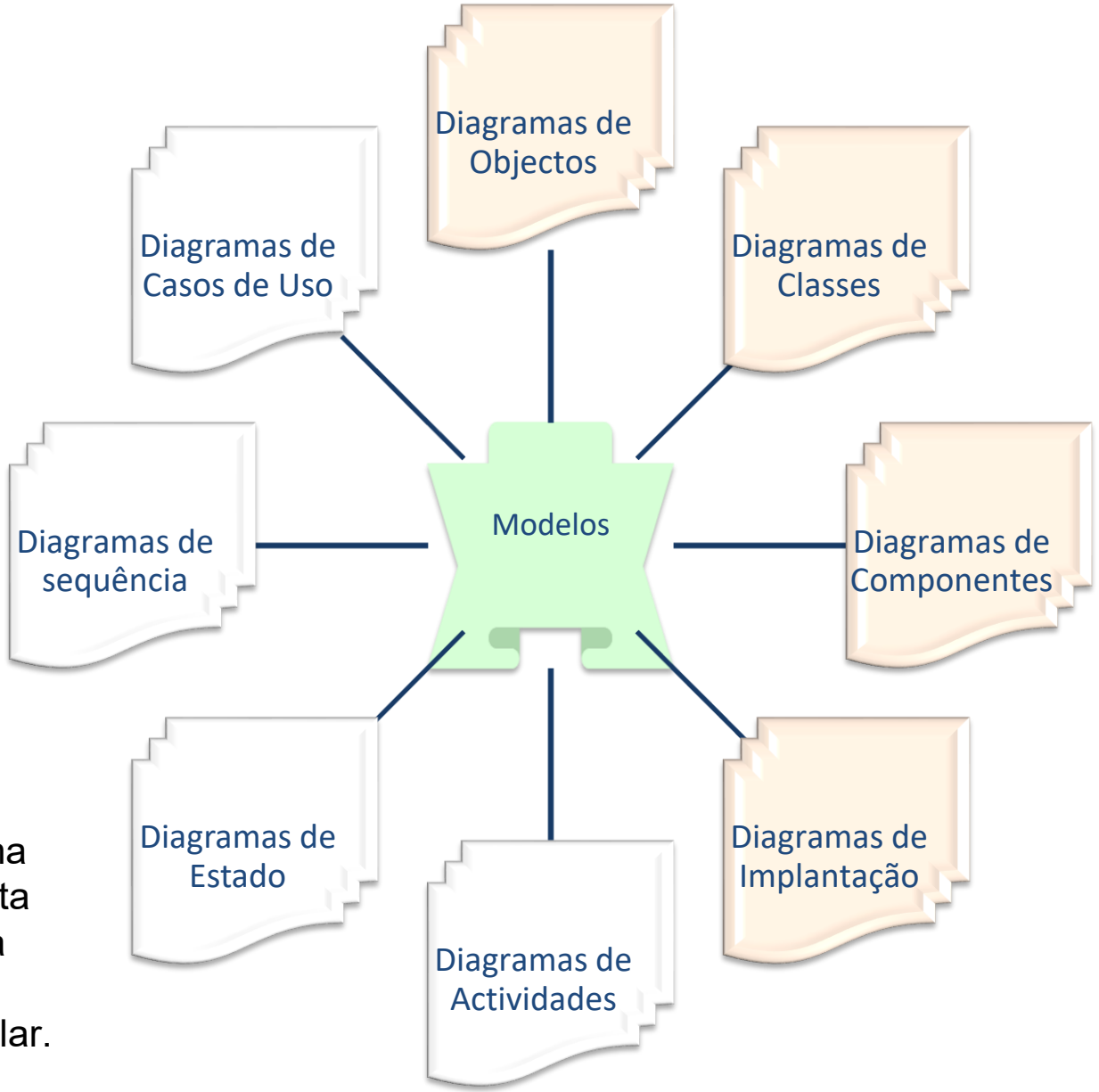
```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet
{
    public void paint (Graphics g)
    {
        g.drawString("Hello, World!", 10, 10);
    }
}
```







Modelos, Diagramas e Visões (1.x)



Um **modelo** é uma descrição completa de um sistema a partir de uma perspectiva particular.

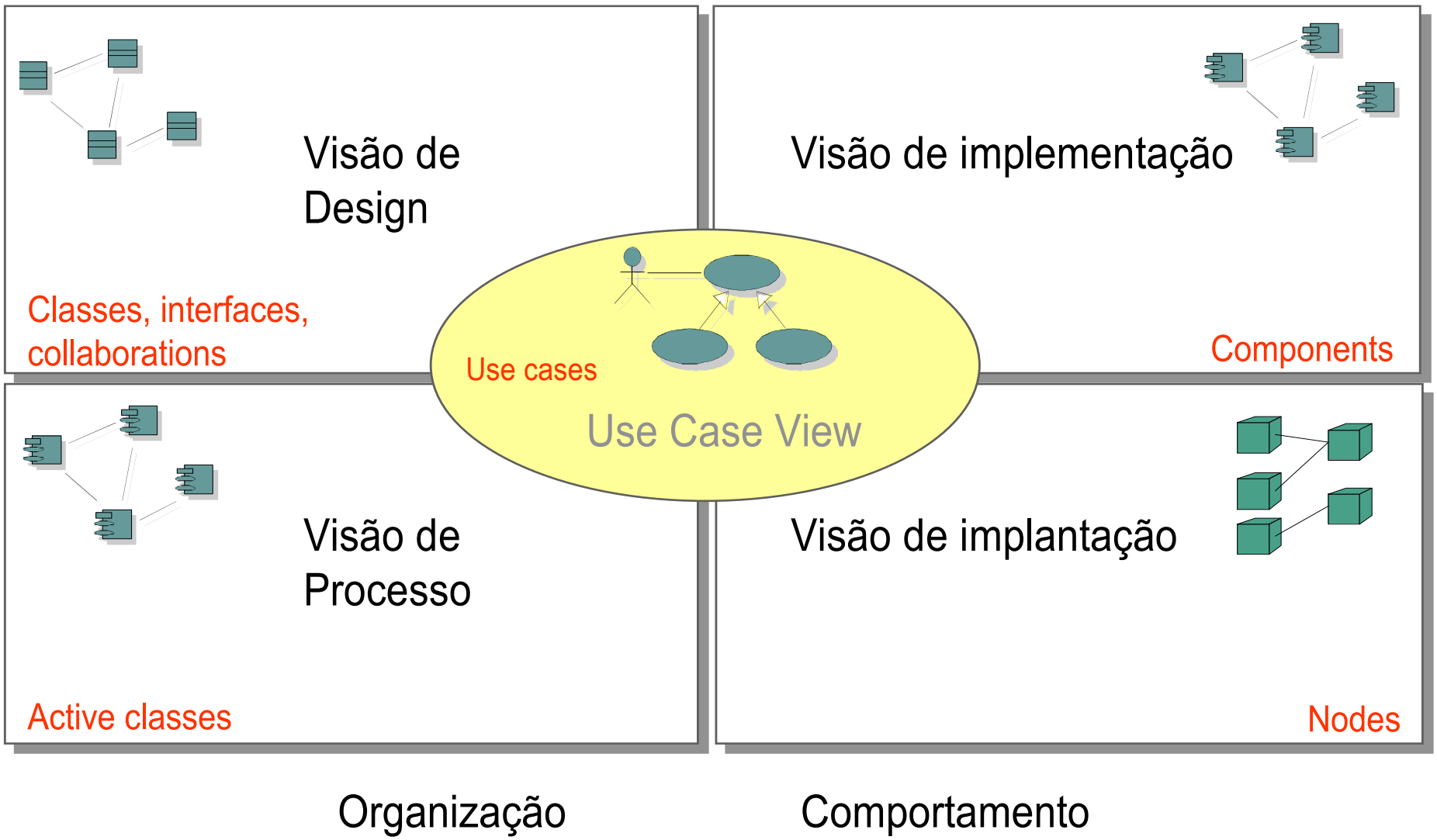
Estes slides estão baseados em material disponível na internet pela Rational Software.
Foi utilizada tradução de João Pascoal Faria, Univ. do Porto

- Um **diagrama** é uma visão sobre um modelo
  - De acordo com o interesse de uma das partes envolvidas (*stakeholder*)
  - Proporciona uma representação parcial do sistema
  - Deve ser semanticamente consistente com outras visões

- Na UML 1.5, há 9 diagramas padrões e 3 de organização:
  - diagramas **de visão estática**: casos de uso (*use case*), classes, objectos, componentes, implantação (*deployment*)
  - diagramas de **visão dinâmica**: sequência, colaboração, estados (*statechart*), actividades
  - diagramas de **organização**: pacotes, subsistemas e modelos

- O mesmo elemento (exemplo: classe) pode aparecer em vários diagramas de um modelo

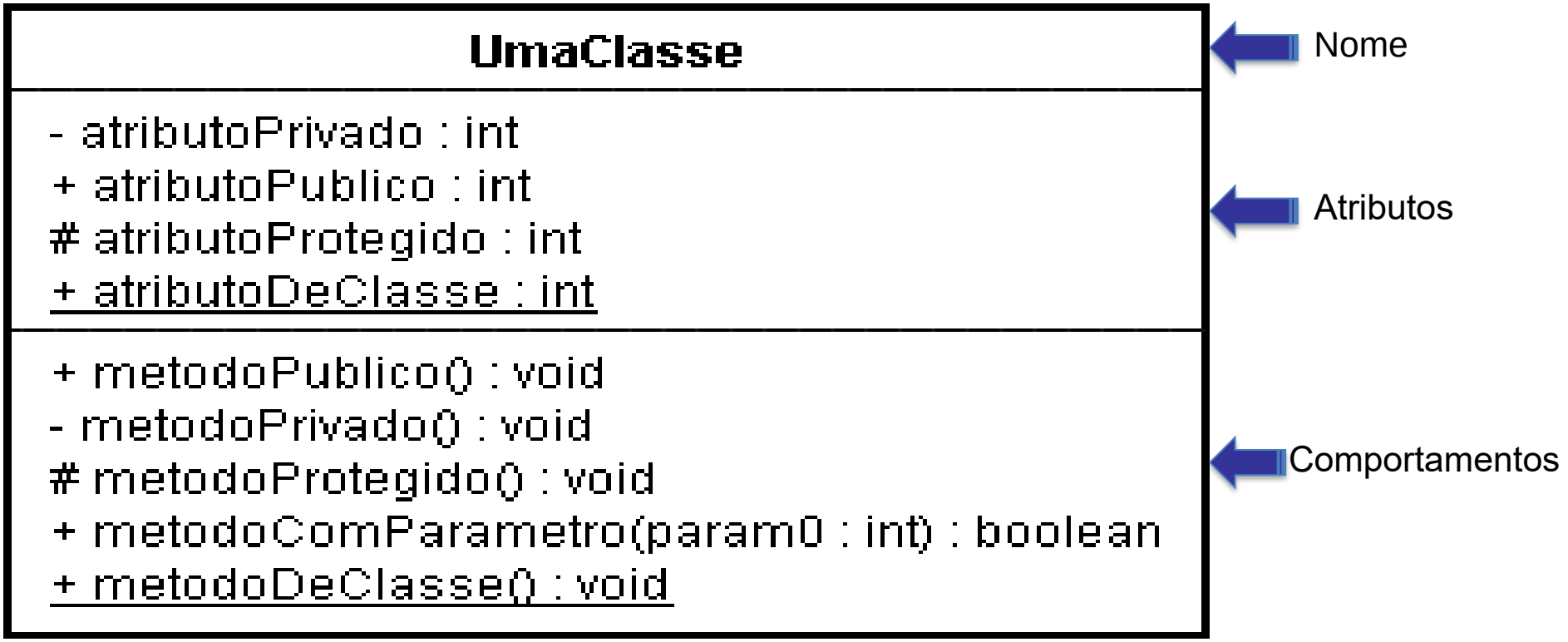
Diferentes Visões



Estes slides estão baseados em material disponível na internet pela Rational Software.
Foi utilizada tradução de João Pascoal Faria, Univ. do Porto

Diagrama de classes

Para melhor visualizar e entender a hierarquia de classes utilizam a notação gráfica. Diagrama de classes denota a estructura estática do sistema. É necessário identificar seus **componentes**, suas **características** e **comportamentos** e o **relacionamento** entre estes componentes.



Exemplo:

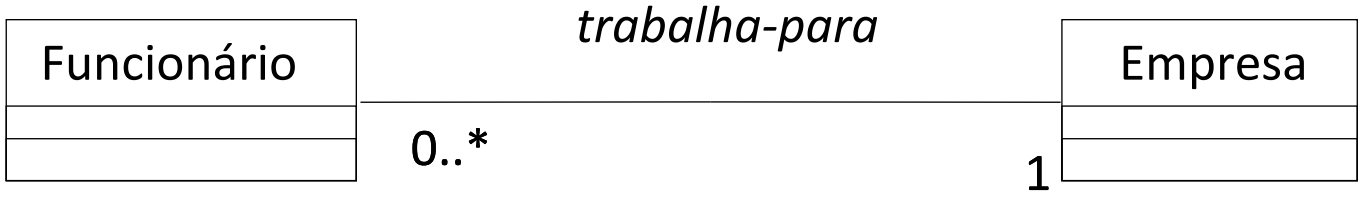
Individuo

+ cont : int
- nome : String
- idade : byte
- endereco : String
- sexo: char

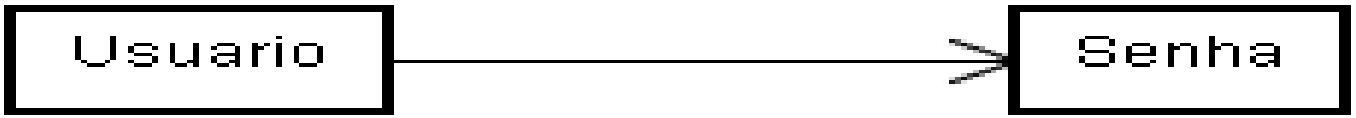
+ Individuo(nome: String, idade: byte, endereco: String, sexo: char)
+ getNome () : String
+ getEndereco() : String
- validarIdade() : byte
+ setEndereco (novoEndereco: String) : void
+ toString() : String

Tipos de relacionamentos básicos

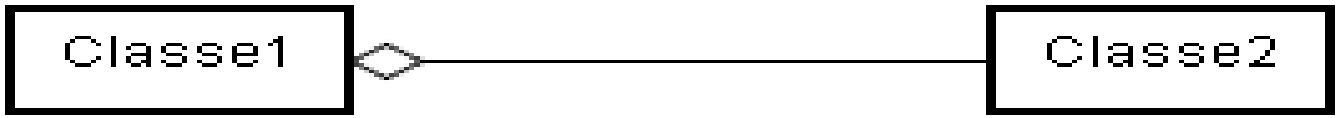
- **Associação**. É um relacionamento estrutural que descreve um conjunto de ligações, onde uma ligação é uma conexão entre objectos



- **Associação direccionada**



- **Agregação.** Representa um relacionamento estrutural “parte de”, “tem”. Uma classe agrega outra se seus objetos contêm objetos da outra classe



Multiplicidade nos relacionamentos

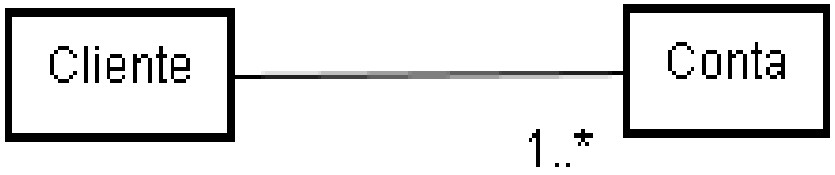
- qualquer número (zero ou mais): *
- um ou mais: 1..*
- zero ou um: 0..1
- exatamente um: 1

Exemplos de Multiplicidade

(1-1): cliente tem uma e somente uma conta

(0-1): cliente pode ter uma conta

(1-N): cliente tem no mínimo um conta, mas pode ter mais



(0-N): cliente pode ter várias contas

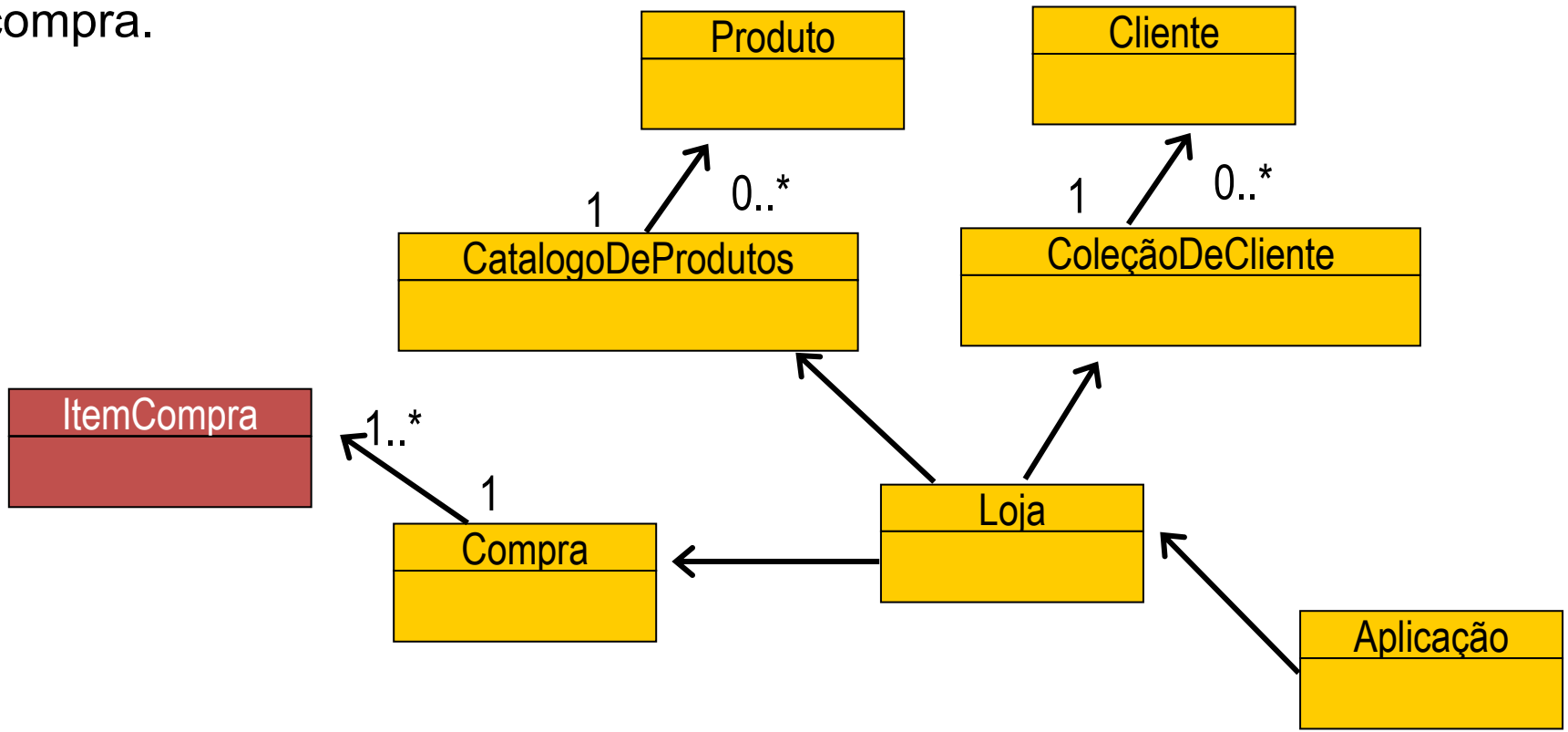
Exemplo de modelagem de sistema

Pretende-se modelar e implementar uma aplicação para uma Loja Virtual. A loja possui catálogo de produtos e coleção de clientes. Cada produto possui código numérico, nome e preço. Cada cliente possui código numérico, nome e endereço.

A loja deverá ser capaz de listar todos produtos do catálogo, bem como todos clientes cadastrados.

Um cliente, ao fazer uma compra, relaciona uma lista de produtos e suas respectivas quantidades.

Ao final da compra, a loja apresenta pedido completo do cliente, incluindo seu nome, lista de produtos e suas quantidades, preços unitários, e total da compra.



Desenho de classes

❖ Caso se pretenda cumprir rigorosamente os requisitos do encapsulamento de objectos numa classe, então todas as suas variáveis internas devem ser declaradas como `private`. O acesso é feito através de métodos selectores (`geters`) e modificadores (`seters`).

❖ Nem todas as variáveis de instância necessitam de selectores e modificadores. Algumas variáveis apenas precisam de ser modificadas na altura da criação de uma instância, não necessitando de ser alteradas ou consultadas posteriormente.

❖ Não usar demasiado tipos básicos numa classe.

❖ Se numa classe são declaradas de forma independente diversas variáveis com uma determinada relação entre si, então é preferível utilizá-las como variáveis de instância de uma nova classe que as agrupe.

Um exemplo típico desta situação é a declaração das variáveis `Rua`, `Número`, `CódigoPostal`, `Localidade`, `Cidade`, `País`, etc. Nesta situação, é preferível exportar toda esta informação para uma nova classe `Morada`, e definir apenas uma variável do tipo `Morada`.

❖ Inicializar sempre explicitamente os dados através de construtores.

```
public class Name {
  private String first, middle, last;

public Name (String firstt, String middlee, String lastt) {
  first = firstt;
  middle = middlee;
  last = lastt;
}
. . .


public class Person {
  private Name nome;
  private char sex;
  private String id;

  public Person (Name nomee, char sexx, String idd) {
  name = nomee;
  sex = sexx;
  id = idd;
}
. . .
public class Tarefas { . . .
  Name n = new Name(“Maria”,”Da”, “Silva”);
  Person p = new Person( n, ‘f’,2345F);
  Person p2= new Person(new Name(“Joze”,”Carlos”, “Pinto”),’m’,23242r);
. . .
```

- Não criar classes com demasiados dados e métodos. Geralmente, a definição de classes muito extensas é reflexo de uma concentração demasiado pesada de funcionalidade numa única classe. É preferível definir classes mais pequenas e mais especializadas.

Existem dois tipos de classes:

- públicas (para utilização geral) e
- auxiliares, que são utilizadas na construção de outras classes.

Em Java, a ordem dos membros é insignificante, no entanto são boas práticas:

- Ordenar os campos consoante as acessibilidades e papéis;
- Organizar os métodos por grupos com a seguinte ordem:
  - Construtores públicos
  - Métodos públicos de acesso ou de selecção (não mudam o estado dos objectos)
  - Métodos públicos de modificação (modificam o estado dos objectos)
  - Construtores não públicos
  - Métodos auxiliares

```
public class ExemploClass {
    <constantes públicas>
    <construtores públicos>
    <métodos de acesso públicos>
    <métodos de modificação públicos>
    <campos não públicos>
    <construtores não públicos>
    <métodos auxiliares não públicos>
    <classes internas>
  }
```

- O método `toString()` deve retornar a representação em `String` do objecto. Deve incluir a representação em *string* de todos os atributos do objecto.

Exercícios

- Ex: Numa turma de N (<30) rapazes e meninas pretende-se saber qual é o mais alto de cada sexo.
- Desenhar o Diagrama de classe utilizando UML
- b) Elabore a classe Aluno, com a informação que considerar necessária, para utilizar num programa que determine o pretendido.
- c) resolver o memso exercíco e e criar classes Menu e Validacoes.

Referência bibliográfica:

António José Mendes; Maria José Marcelino.
"Fundamentos de programação em Java 2". FCA. 2002.

Elliot Koffman; Ursula Wolz.
"Problem Solving with Java". 1999.

John R. Hubbard.
"Theory and problems of programming with Java". Schaum's Outline series. McGraw-Hill.

H. Deitel; P. Deitel.
"Java, como programar". 4 edição. 2003. Bookman.

Rui Rossi dos Santos.
"*Programando em Java 2– Teoria e aplicações"*. Axcel Books. 2004