Programação Orientada a Objetos

Aula 03: Criação de classes e objetos

Ivo Calado

ivo.calado@ifal.edu.br

Instituto Federal de Educação, Ciência e Tecnologia de Alagoas

24 de Agosto de 2016

Roteiro

1. Classes
2. Arrays e ArrayLists
3. Associações
4. Pacotes e Encapsulamento
5. Um pouco mais sobre this

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Definição I

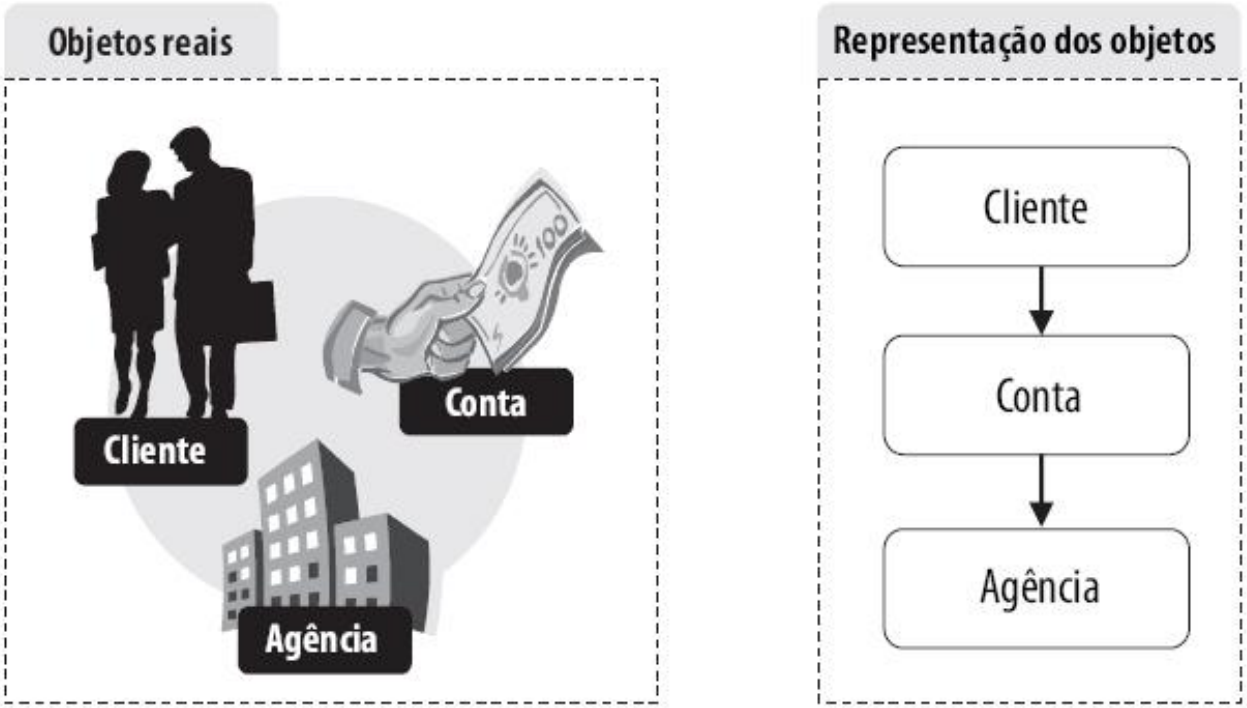
Definição

POO é um paradigma de programação onde um programa consiste de uma rede de objetos se comunicando

- O programa passa a ser composto por um conjunto de **entidades** que se comunicam
- Cada **entidade** é chamada de **objeto** no domínio de POO

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Definição II



Definição III

Figura: Linguagens OO oferecem suporte explícito para representar objetos do mundo real em software

Definição IV

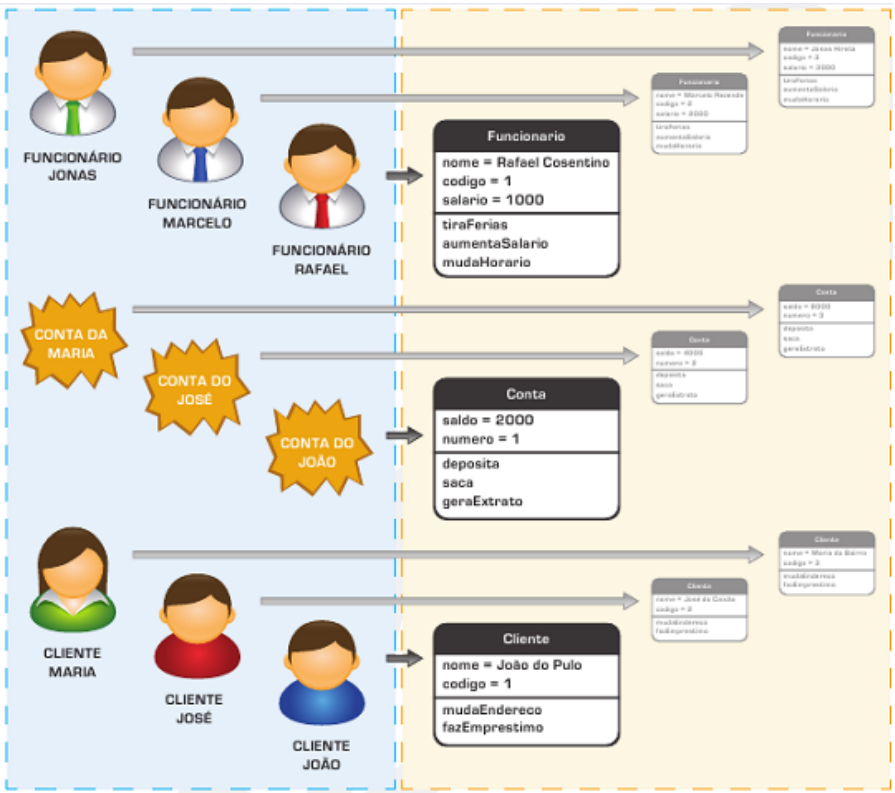


Figura: Objetos reais mapeados para objetos no paradigma OO

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Mas como podemos representar um objeto?

Linguagens OO geralmente oferecem dois recursos para representarmos objetos, que são **atributos** e **métodos**

- **Atributos:** armazenam dados do objetos. Também conhecidos como estados do objetos
- **Métodos:** correspondem as **funções** do objeto
  - Podem manipular os valores dos atributos
  - Possbilitam interagir com outros objetos
  - Uma requisição de execução de um método é chamada **mensagem**

Mas como podemos representar um objeto?

Exemplo 1: como poderíamos representar modelar uma lâmpada em termos de atributos e métodos?

- **Atributos:**
  - EstaLigado: [Verdadeiro/Falso]
- **Métodos:**
  - Ligar
  - Desligar

Mas como podemos representar um objeto?

Exemplo 2: como poderíamos representar modelar uma lâmpada com suporte a ajuste de potência, em termos de **atributos** e **métodos**?

Mas como podemos representar um objeto?

Exemplo 2: como poderíamos representar modelar uma lâmpada com suporte a ajuste de potência, em termos de **atributos** e **métodos**?

- **Atributos:**
  - EstaLigado: [Verdadeiro/Falso]
  - PotênciaAtual: [*Watts*]
- **Métodos:**
  - Ligar
  - Desligar
  - AjustarPotência(novaPotência)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Mas como podemos representar um objeto?

Qual a diferença entre uma função na programação estruturada e um método de um objeto na OO?

Funções na programação estruturada não estão associadas a nenhum objeto enquanto que métodos **pertencem** a um objeto

Quais informações modelar sobre um objeto?

Suponha que fosse solicitado a modelagem de uma pessoa. Quais atributos deveriam ser considerados?

Quais informações modelar sobre um objeto?

Suponha que fosse solicitado a modelagem de uma pessoa. Quais atributos deveriam ser considerados?
Algumas propriedades "óbvias" poderiam ser:

- 1 - nome
- 2 - endereco
- 3 - cpf
- 4 - dataNascimento

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Quais informações modelar sobre um objeto?

Porém, poderíamos considerar algumas outras propriedades, como:

- 5 - peso
- 6 - altura
- 7 - cor dos olhos
- 8 - cor dos cabelos
- 9 - dias que não toma banho
- 10 - tamanho da unha do dedão do pé

Quais propriedades deveríamos considerar?

Quais informações modelar sobre um objeto?

Porém, poderíamos considerar algumas outras propriedades, como:

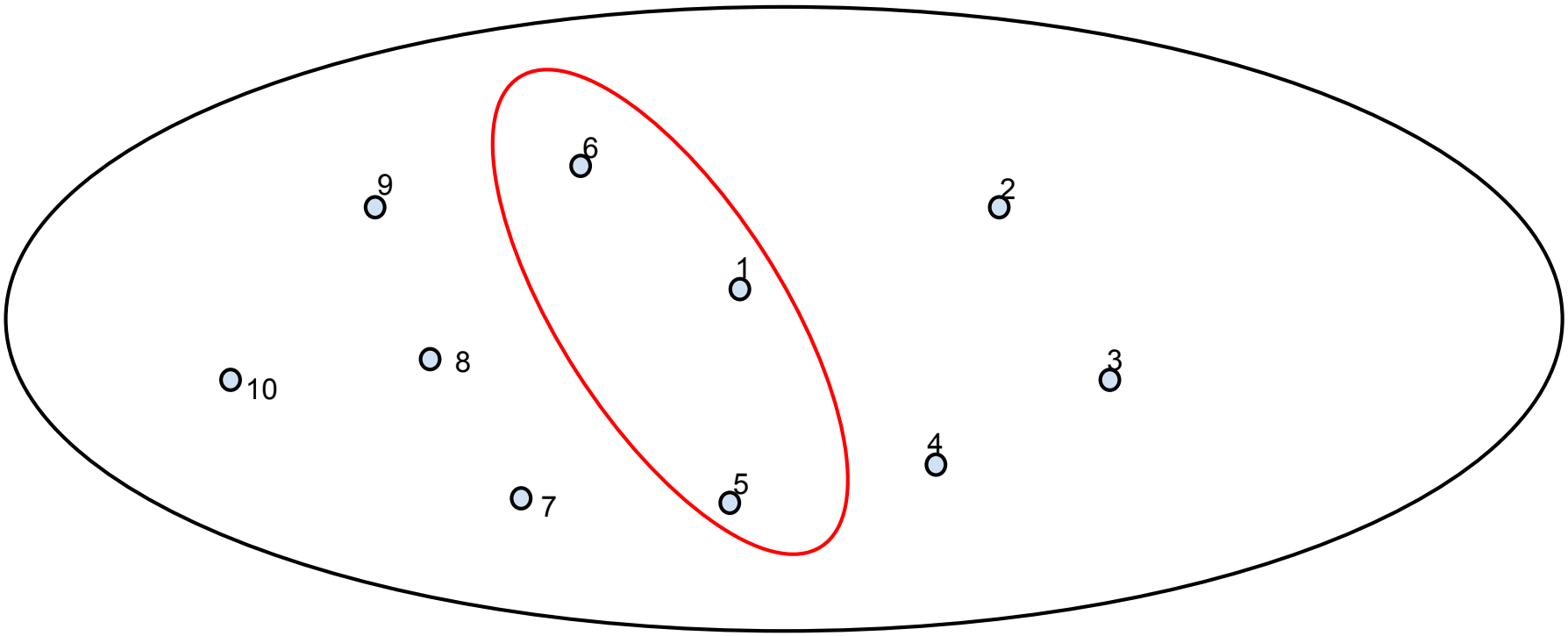
- 5 - peso
- 6 - altura
- 7 - cor dos olhos
- 8 - cor dos cabelos
- 9 - dias que não toma banho
- 10 - tamanho da unha do dedão do pé

Quais propriedades deveríamos considerar?

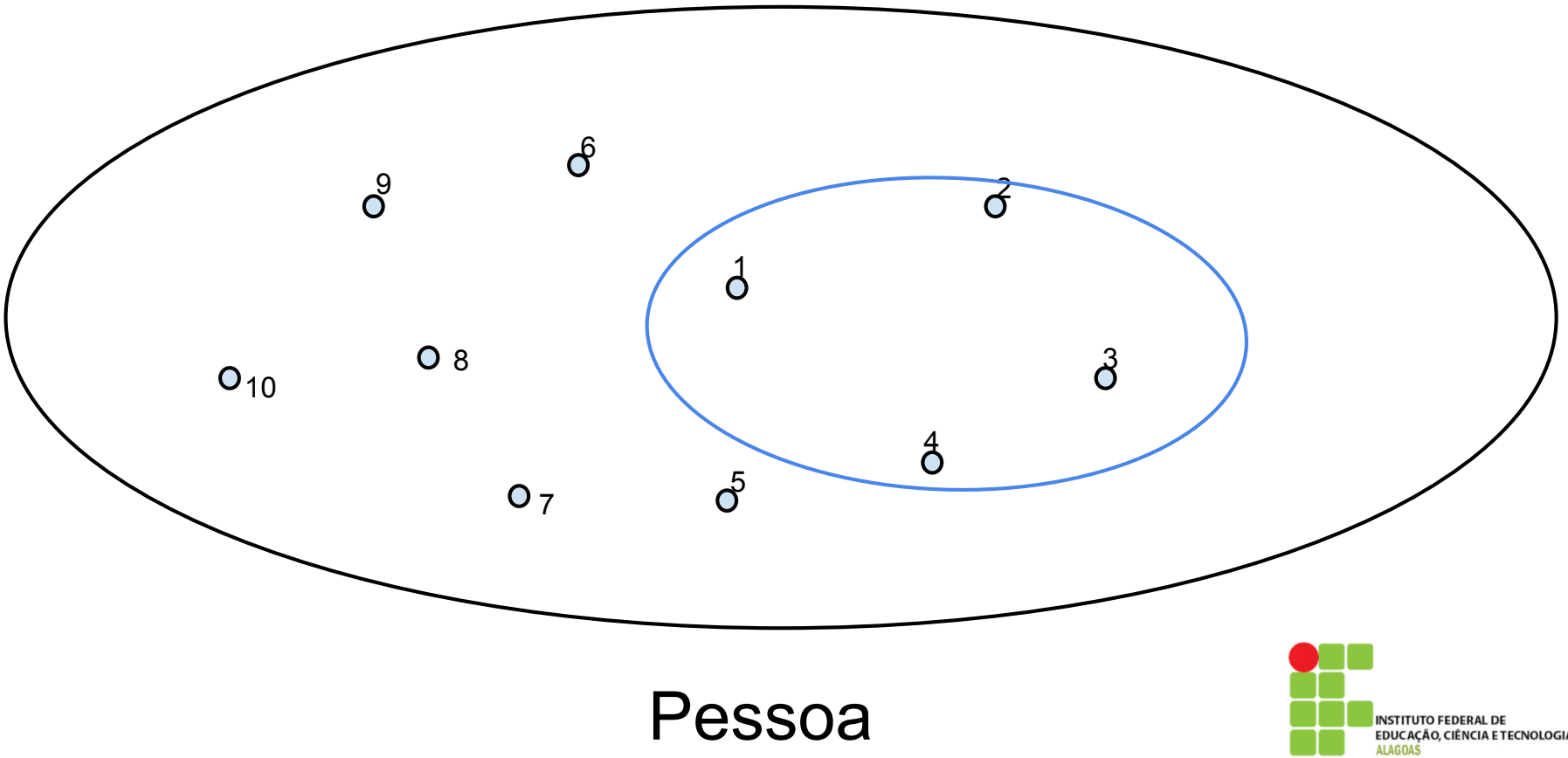
Depende do **domínio** do problema!

Abstraindo atributos I

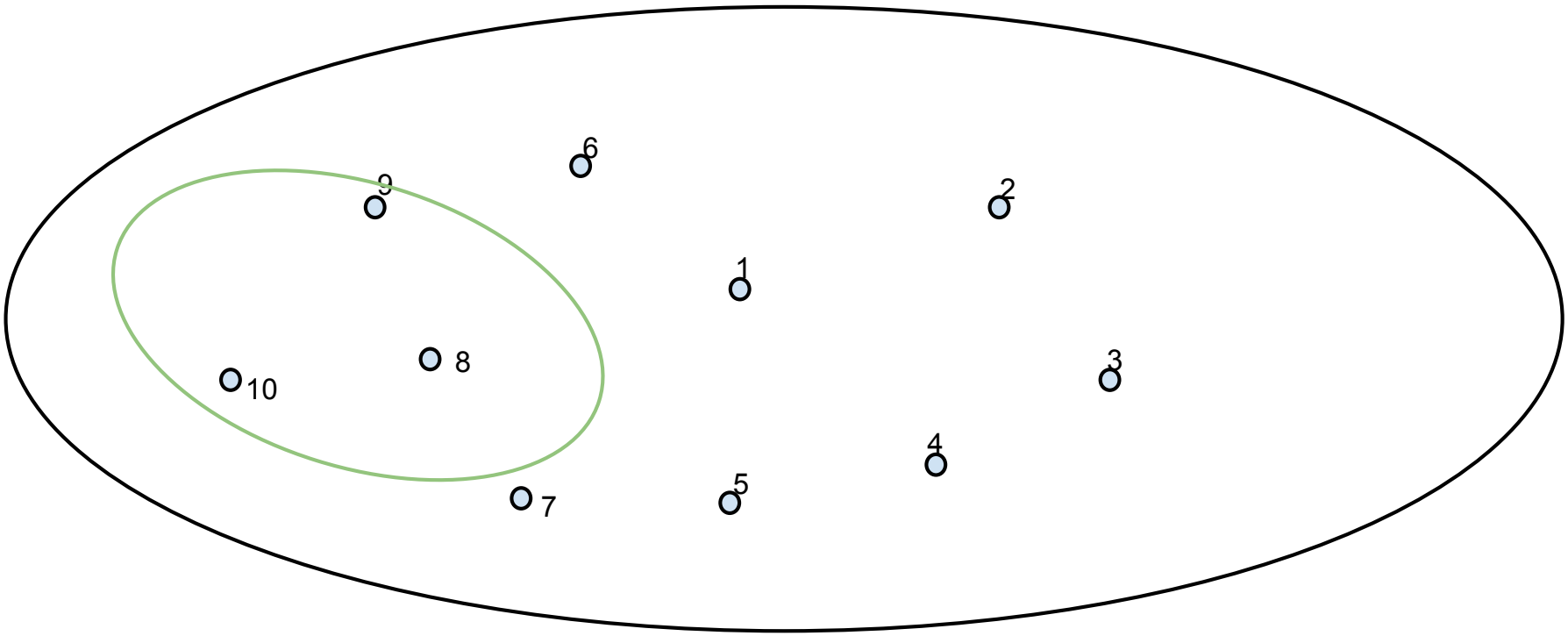
Abstraindo atributos II



Abstraindo atributos III

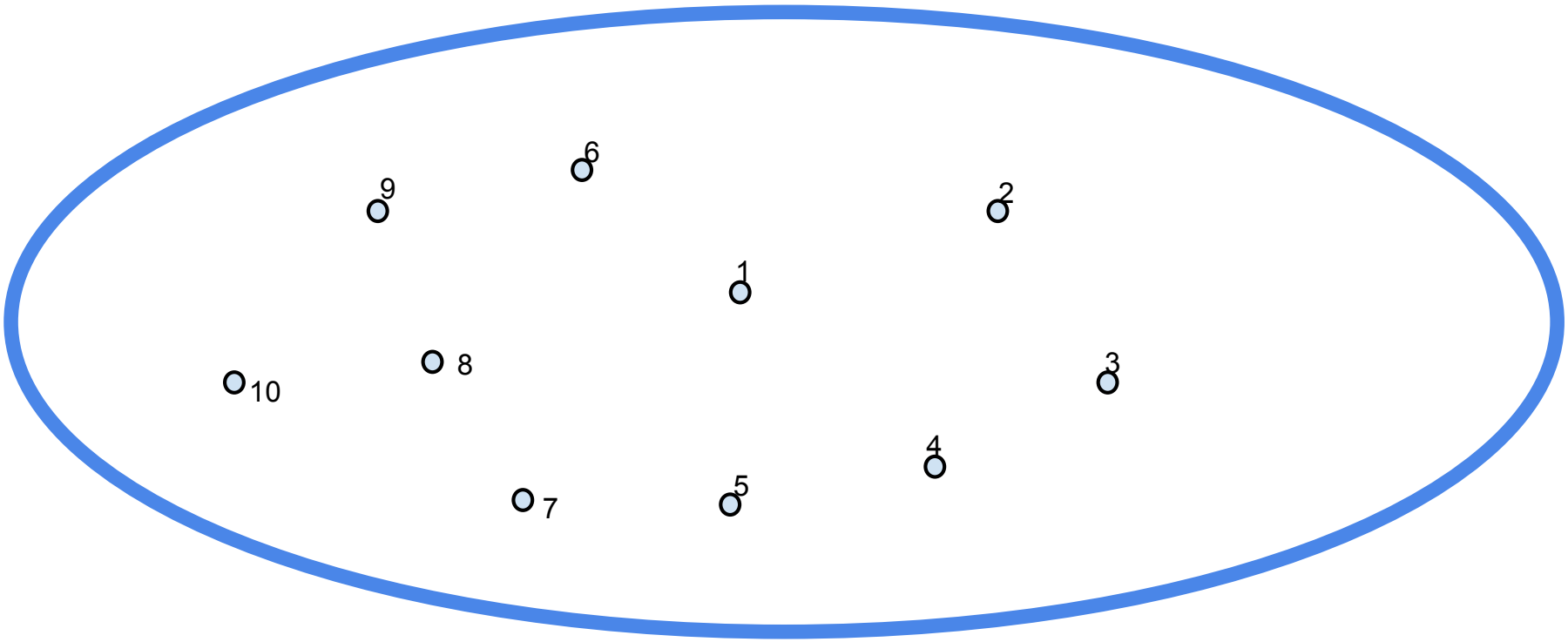


Abstraindo atributos IV



Pessoa

Abstraindo atributos V



Seleção das propriedades importantes = Abstração

Abstração

"A **abstração** *é o processo de filtragem de detalhes sem importância do objeto, para que apenas as* **características apropriadas** *que o descrevem permaneçam"*

Seleção das propriedades importantes = Abstração

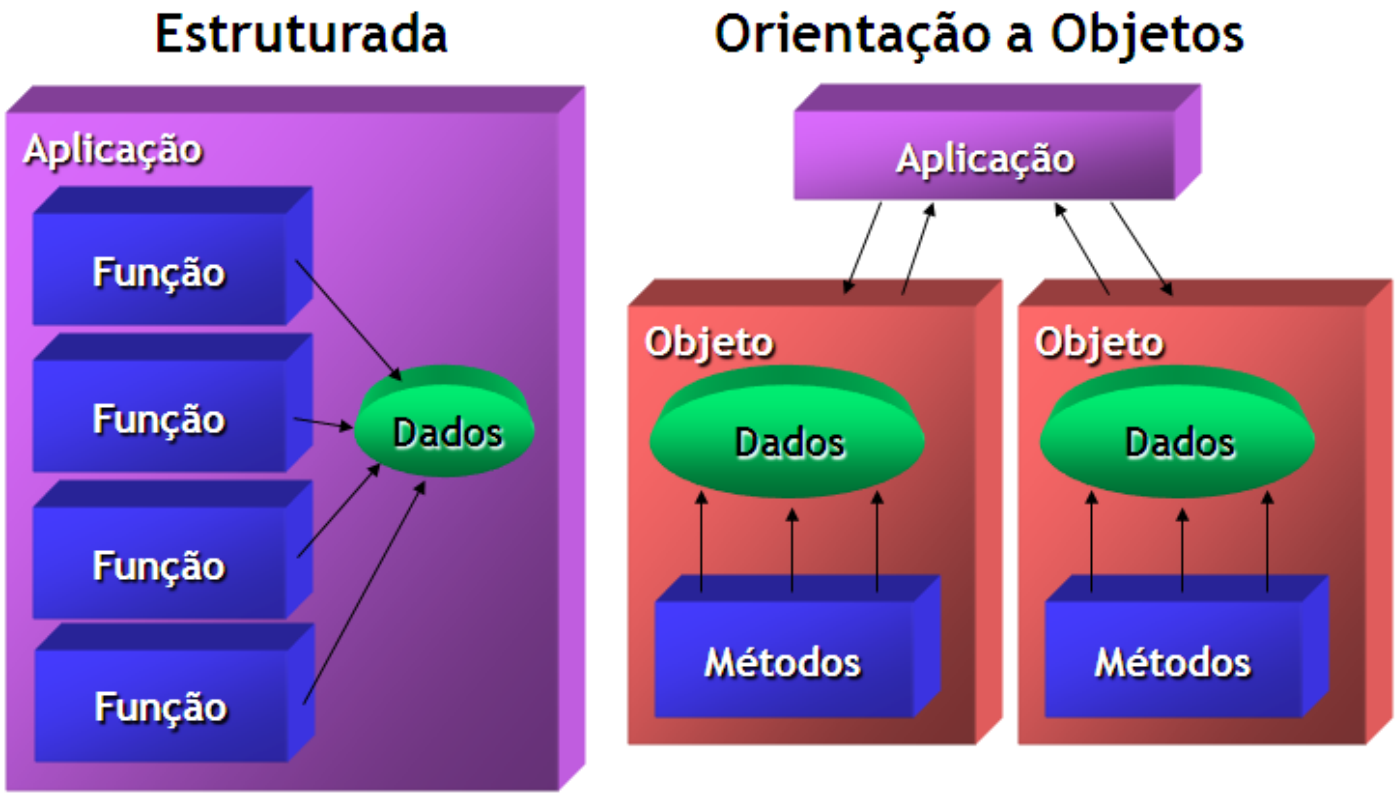
Abstração

"A **abstração** *é o processo de filtragem de detalhes sem importância do objeto, para que apenas as* **características apropriadas** *que o descrevem permaneçam"*

- As **características apropriadas** sempre dependem do **domínio** que está sendo trabalhado

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

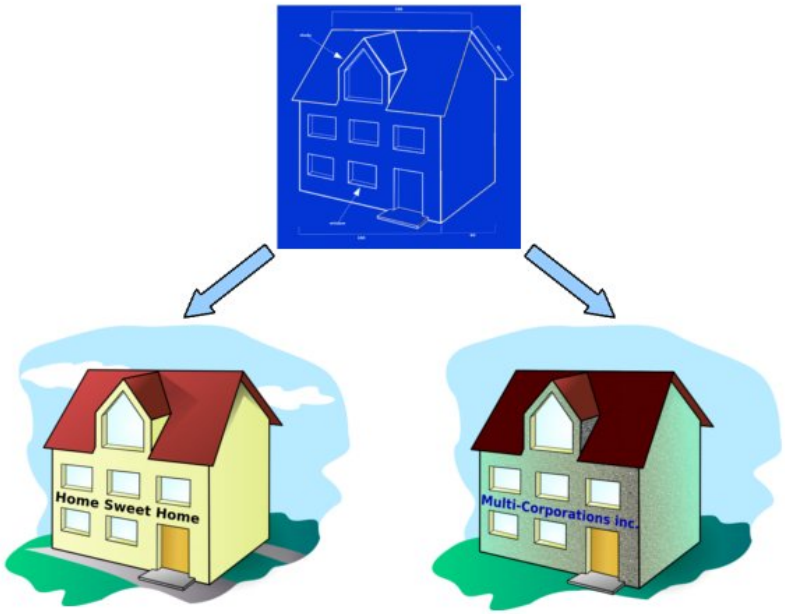
Resumo: POO x Programação Estruturada



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Classes I

- Antes de um objeto ser criado devemos projetá-lo



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Classes II

- Cada classe deve pode ter um conjunto de **atributos** e **métodos**
  - Atributos representam as **propriedades** do tipo a ser criado
  - Métodos representam as **ações** que o tipo pode realizar
- Um objeto é a "concretização de uma classe" **Um objeto é uma instância de uma classe**

Fazendo uma analogia com uma tabela

- A **tabela** seria a **classe**
- Cada linha da tabela (**tupla**) seria um **objeto**

- Todas as instâncias de uma classe têm os mesmos métodos e atributos mas podem ter valores diferentes

Criação de classes

Definição da classe:

```
public class Pessoa {
  /*Neste ponto são inseridos métodos e atributos*/
}
```

Instanciação da classe:

```
Pessoa p = new Pessoa();
```

Atributos

- Os atributos representam propriedades da classe.
- Podem ser tipos primitivos ou complexos definidos

```
public class Pessoa {
    String nome;
    int idade;
    Endereco endereco;
    private String cpf;
}
```

Utilização:

```
Pessoa p = new Pessoa();
p.nome = "Maria";
p.cpf= "0011"; //Erro! Não é possível acessar
    diretamente atributos privados
```

Métodos

- Métodos representam ações do objeto
- Podem receber parâmetros ou não e podem ter retorno ou não

```
public class Funcionario {
  double salario;
  public void adicionarBonificacao(double bonificacao
      ) {
    salario += bonificacao;
  }
}
```

Utilização:

```
Funcionario f = new Funcionario();
f.salario = 1000;
f.adicionarBonificacao(500);
```

Tipos de retorno de métodos

- Cada método deve definir o tipo de retorno ou informar que o método não irá retornar nada com a palavra-chave **void**
- Métodos que têm retorno utilizam a palavra-chave **return** para indicar o valor retornado, que obrigatoriamente deverá ser do tipo especificado na assinatura do método

```
...
public double calcularSalarioFinal() {
  return salario*1.25;
}
...
```

Sobrecarca de métodos

- É possível definir dois ou mais métodos com o mesmo nome mas com assinaturas distintas
- A JVM irá diferenciar o método correto a ser chamado a partir dos tipos passados

```
...
 public void adicionarBonificacao(double bonificacao)
     {...}

 public void adicionarBonificacao(double bonificacao1,
      bonificacao2) {...}
...
```

Construtores I

- Construtores representam um tipo especial de métodos que são chamados implicitamente pela JVM durante o processo de instanciação de um novo objeto
- Diferentemente dos métodos tradicionais, os construtores não definem tipo de retorno (nem mesmo **void**)
- Outra particularidade é que os construtores devem ter **obrigatoriamente** o mesmo nome da classe

Construtores II

```
public class Pessoa {
  ... //Definição de atributos e métodos
  public Pessoa() {...}

  public Pessoa(String nome) {...}

  public Pessoa(String nome, String endereco) {...}
}
```

Utilização:

```
Pessoa p1 = new Pessoa();
Pessoa p2 = new Pessoa("maria");
Pessoa p3 = new Pessoa("maria", "Palmeira dos indios")
    ;
```

Construtores III

- Por padrão, quando não definido, toda classe possui um construtor **default** que é um construtor sem parâmetros
- Todavia, caso seja definido um construtor específico, o construtor **default** só fica disponível SE for definido explícitamente

```
public class Pessoa {
... //Definição de atributos e métodos

  public Pessoa(String nome){...}
}
```

Utilização:

Construtores IV

```
Pessoa p1 = new Pessoa("Maria"); //Ok, foi definido um
    construtor com essa assinatura
Pessoa p2 = new Pessoa();// ERRO! Como foi definido um
    construtor não-default, o construtor default so
   fica disponível se definido explicitamente
```

Definição de Arrays I

- Conforme discutido anteriormente, um array representa uma sequência continua de elementos **do mesmo tipo**
- Para criação de um vetor, deve-de definir o tamanho do vetor
- É importante lembrar que a primeira posição de um vetor inicia em **0**

```
int vet[] = new int[10];
vet[0] = 3; //Modifica a primeira posição do vetor
vet[9] = 9; //Modifica a última posição do vetor
vet[10] = 0; //Erro!
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Definição de Arrays II

- Diferentemente dos tipos primitivos, para tipos complexos é necessário inicializar cada posição do vetor

```
Pessoa vet[] = new Pessoa[10];
vet[0] = new Pessoa(); //Inicialização
vet[0].setNome("maria"); //Ok!
vet[1].setNome("João"); //Erro! NullPoiterException
```

ArrayList I

- Arrays em Java são estruturas de dados cujos tamanho são definidas antes da estrutura ser usada
- Todavia, em algumas situações não é possível precisar qual será o tamanho inicial dos elementos a serem inseridos
- Neste sentido, é possível fazer uso da classe **ArrayList** para criação de listas de objetos

ArrayList II

```
import java.util.ArrayList;
...

ArrayList<Pessoa> lista = new ArrayList<>(); //Cria a
    lista
Pessoa p = new Pessoa();
lista.add(p); //Adiciona o elemento à lista
int tamanho = lista.size(); //Retorna o tamanho atual
    da lista
Pessoa p2 = lista.get(0); //Recupera a primeira
    posição da lista
```

- Durante a criação da lista, deve-se definir qual será o tipo do elemento a ser inserido

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

ArrayList III

- O tipo do elemento obrigatoriamente deve ser um tipo complexos
- A utilização de tipos primitivos gera erro sendo necessária a utilização de classes *wrappers*
- A conversão entre o tipo primitivo e *wrapper* correspondente é feita de maneira automática e transparente desenvolvedor

Tipo primitivo	**Classe wrapper correspondente**
boolean	Boolean
short	Short
int	Integer
long	Long
float	Float
double	Double

ArrayList IV

Tipo primitivo	Classe wrapper correspondente
char	Character
byte	Byte

Exemplo:

```
int a = 10;
Integer b = a;
int c = b;

ArrayList<Integer> lista = new ArrayList<>();
lista.add(a);
lista.add(b);
lista.add(c);
```

Principais métodos da lista? I

Método	Descrição
add(E elemento)	Adiciona o objeto E no final da lista
add(int index, E element)	Adiciona o objeto E na lista, na posição indicada pelo parâmetro index
isEmpty()	Retorna um boolean, indicando se a lista está vazia
size()	Retorna um int indicando o tamanho da lista
remove(int index)	Remove o objeto (não interessa qual) no indice indicado pelo parametro index

Principais métodos da lista? II

Método	Descrição
remove(E elemento)	Remove da lista o objeto indicado pelo parâmetro o. Esse objeto deve implementar o método *equals*
indexOf(E elemento)	Retorna o índice da **primeira** ocorrência do elemento passado ou -1 caso não seja encontrado
lastIndexOf(E elemento)	Retorna o índice da **última** ocorrência do elemento passado ou -1 caso não seja encontrado

Como iterar na lista I

- Para iterar na lista é possível utilizar a estrutura for de duas formas

Forma 1 (acesso ao índice da posição):

```
ArrayList<Integer> lista = new ArrayList<Integer>();
... //Adições e remoções elementos

for(int i = 0; i < lista.size(); i++) {
  int lista.get(i); //Recupera a i-ésima posição da
      lista
}
```

Forma 2 (sem acesso ao índice da posição):

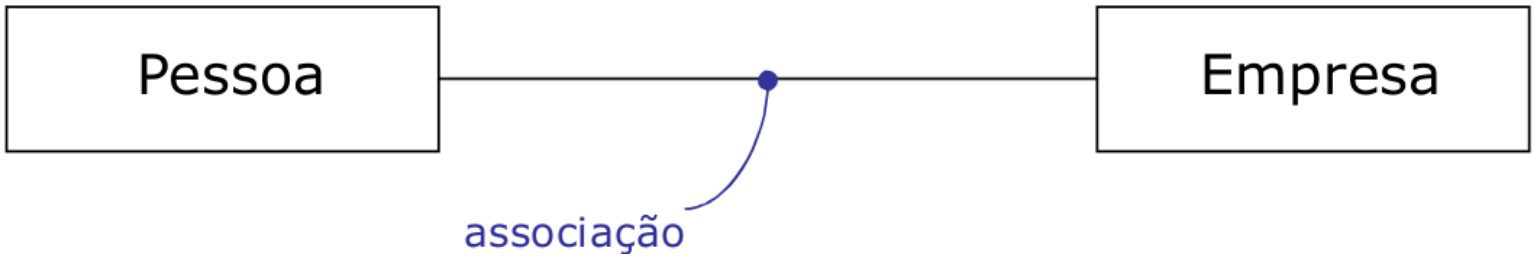
Como iterar na lista II

```
ArrayList<Integer> lista = new ArrayList<Integer>();
 ... //Adições e remoções elementos

 for(Integer v : lista) {
   //a variável 'v' passa a conter o valor atual da
       lista
 }
```

Associação

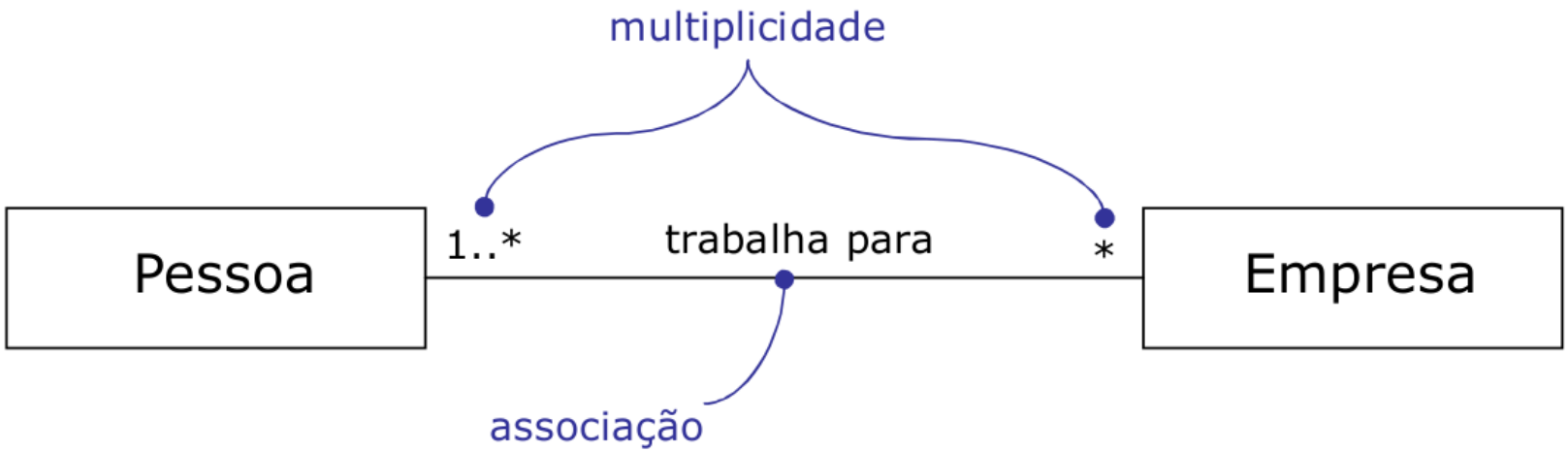
- Uma **associação** é um relacionamento estrutural que indica que os objetos de uma classe estão vinculados a objetos de outra classe.
- Uma associação é representada por uma linha sólida conectando duas classes.



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

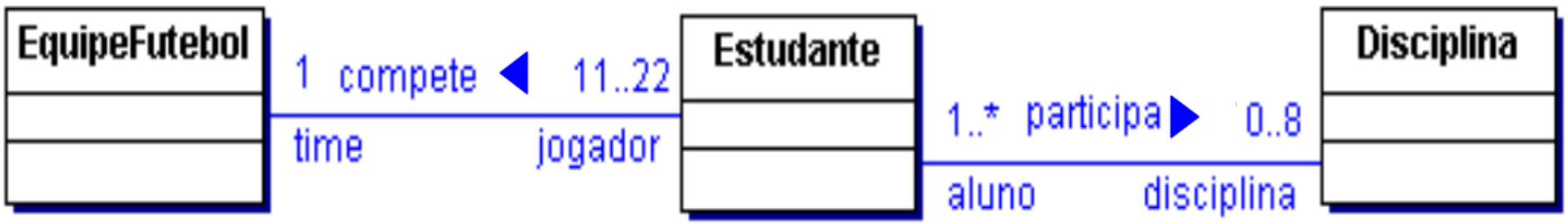
Indicadores de multiplicidade

- 1 => Exatamente um
- 1..* => Um ou mais
- 0..* => Zero ou mais (muitos)
- * => Zero ou mais (muitos)
- 0..1 => Zero ou um
- m..n => Faixa de valores (por exemplo: 4..7)



Exemplo de associação

- Um **Estudande** pode ser
  - um **aluno** de uma Disciplina
  - um **jogador** da Equipe de Futebol
- Cada Disciplina deve ser cursara por no mínimo 1 aluno
- Um aluno pode cursar de 0 até 8 disciplinas



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Agregação

- Trata-se de um tipo especial de associação
- Utilizada para indicar relacionamento "todo-parte"

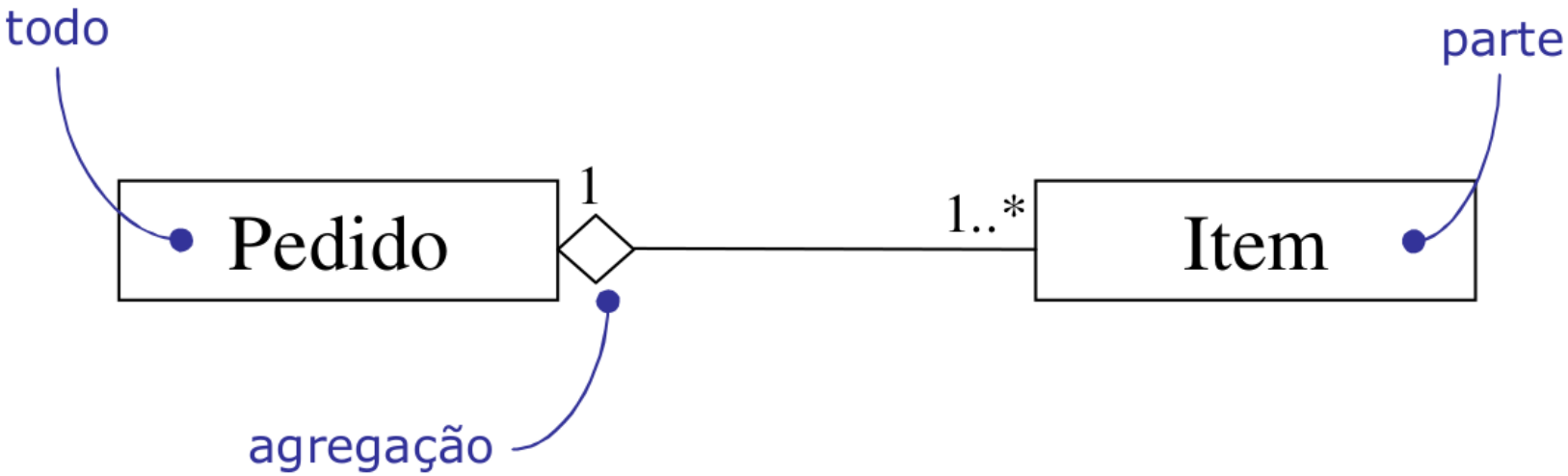


Figura: Um objeto "parte" pode fazer parte de vários objetos "todo"

Composição

- É uma variante semanticamente mais "forte" da agregação
- Os objetos "parte" só podem pertender a um único objeto "todo" e têm o seu tempo de vida coincidente com o dele

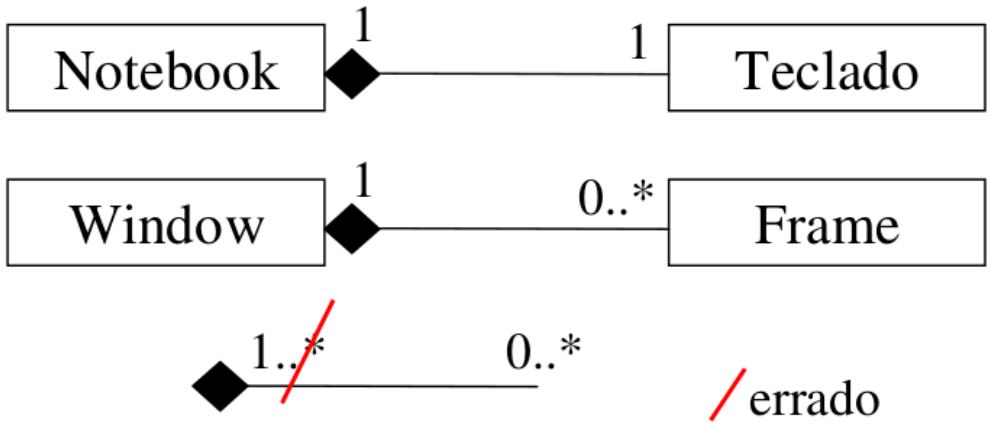
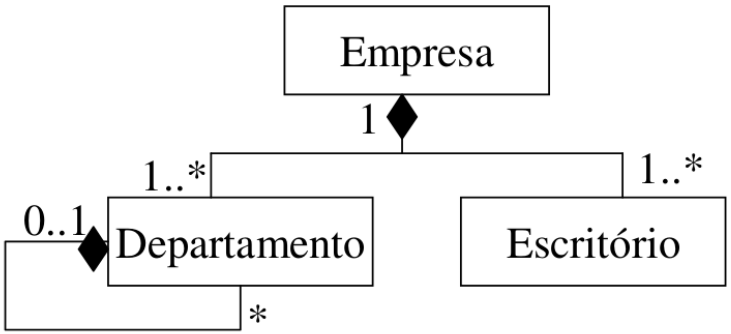
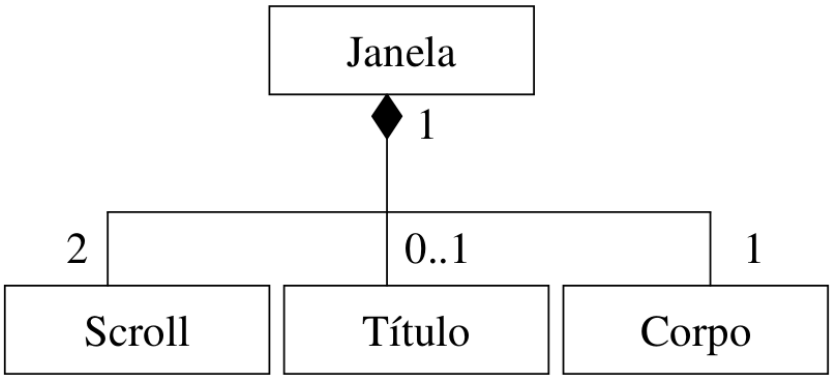


Figura: Quando o "todo" *morre* todas as suas "partes" também *morrem*

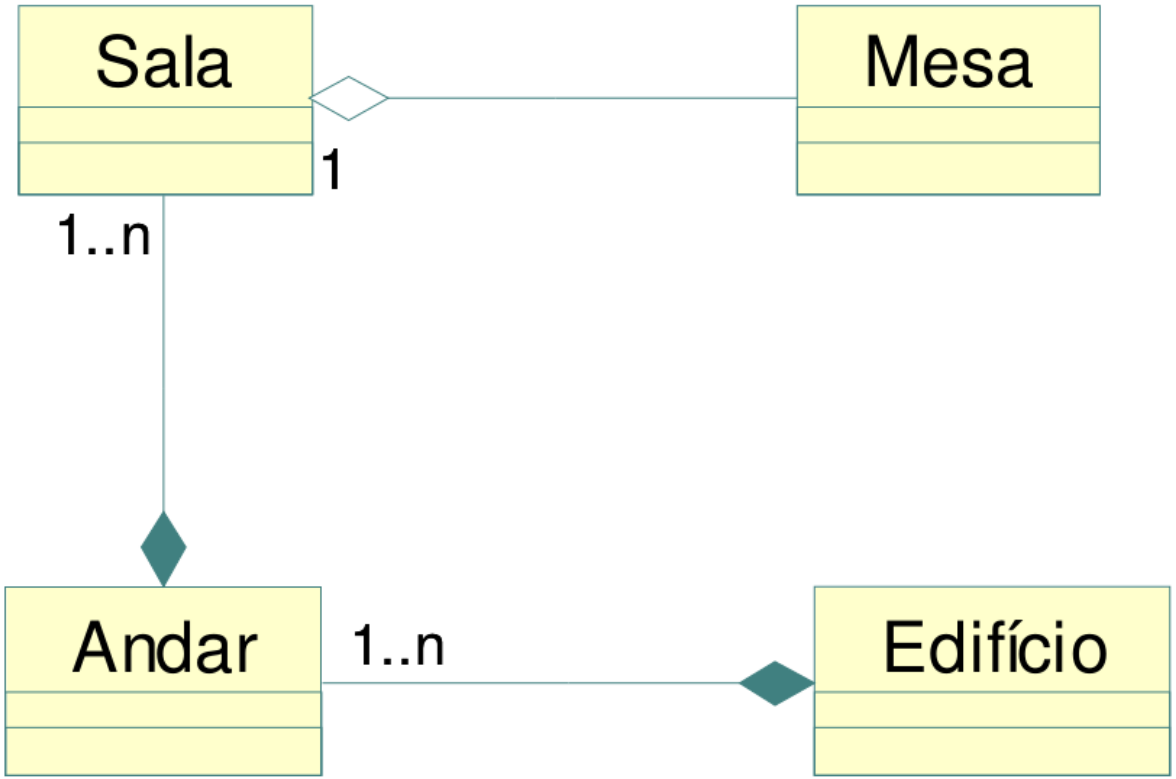
Exemplos de composição





INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

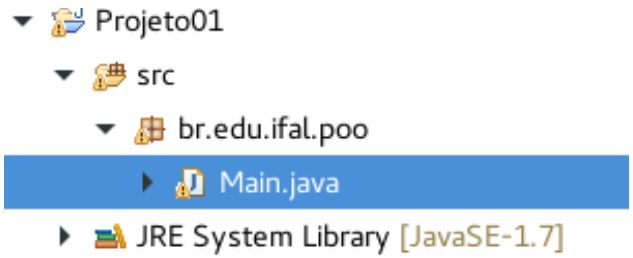
Agregação x Composição



Pacotes I

- Em projetos mais complexos pode surgir a necessidade de construir mais do que uma classe com o mesmo nome mas propósitos distintos. Como nomear as classes?
- Além disso, muitas vezes é interessante agrupar as classes de acordo com a funcionalidade
- Para tal, a linguagem Java fornece o conceito de "pacotes"
- A utiização de pacotes é especialmente útil na construção de bibliotecas
- Com isso, os códigos fontes do Java passam a ser organizados numa hierarquia de pacotes, podendo ser importados a partir da palavra chave **import**

Pacotes II



O que muda na classe?

```
package br.edu.ifal.poo;


public class Pessoa {
        private String nome;
        private String endereco;
        //outros métodos e atributos
}
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Pacotes III

- Uma classe mais interna **não** pode importar uma classe mais externa

Existe alguma regra para nomeação de pacotes?

Utilizar o nome de domínio invertido!

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

O que é encapsulamento?

- Trata-se de uma forma de atribuir diferentes níveis de acesso aos métodos e atributos
- A ideia básica é esconder detalhes da implementação do mundo exterior sempre que possível

Considere o seguinte código:

```
public class Empregado {
    String nome;
    Empregado(String n) {
       nome = n;
    }
}
```

Como impedir que o nome do empregado seja modificado após a construção do objeto?

Níveis de encapsulamento

Define-se 4 tipos de encapsulamento:

- **public**: pode ser acessado de qualquer ponto da aplicação
- **private**: só pode ser acessado internamento
- **protected**: só pode ser acessado na hierarquia de classes (mais detalhes quando formos ver herança!)
- **default**: só pode ser acesso de dentro do próprio pacote

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Regras gerais de utilização

- **Normalmente** atributos são **privados**
- **Normalmente** métodos são públicos
- Atributos e métodos que só tem interesse dentro do pacote devem utilizar encapsulamento **default**

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Qual o efeito do encapsulamento para as classes?

É possível modificar o encapsulamento de uma classe. Todavia segue regras levemente diferentes:

- Em cada unidade de compilação (arquivo .java) **apenas uma única** classe pode ser **pública**
- Classes privadas e protegidas só podem ser definidas internamente

```
public class Pessoa {
  private class Endereco {
    public String cidade; }
  Endereco end;
  public Pessoa() {
    end = new Endereco();
    end.rua = "Palmeira";    }
}
```

Os métodos get e set

- Embora não seja obrigatório, convenciona-se chamar os métodos de acesso/modificação dos atributos de **get** e **set**, respetivamente

Exemplo:

```
//...Código
private String nome;
private String endereco;
public void setNome(String n){nome = n;}
public String getEndereco() {return nome;}
public void setEndereco(String end) {endereco = end;}
public String getEndereco() {return end;}
//...Código
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Um pouco mais sobre métodos set e get: JavaBeans

- A utilização dos métodos set e get é importante visto que diversas bibliotecas de Java levam em consideração essa padronização

O que seriam JavaBean?

Trata-se de uma especificação de software definida pela Sun/Oracle para criação de componentes reutilizáveis. Em termos práticos, um **bean** é uma classe Java com as seguintes propriedades:

- Possuir construtor *default* (isto é, sem argumentos)
- Possuir métodos get e set para os atributos da classe (apenas dos que necessitarem ser acessados).

Atividade para casa

Pesquisar para que serve os modificadores **static** e **final** quando aplicados a métodos, atributos e classes

Utilizando o this para invocar métodos

- Além de referenciar atributos do objeto é possível utilizar o **this** para referenciar um método

```
public class Pessoa {
  public void met1() {}

  public void met2() {
    this.met1();
  }
}
```

Como um construtor pode invocar outro construtor

Considere o seguinte código:

```
public class Pessoa { //definição dos atributos e
    métodos
  public Pessoa(String nome) {
    this.nome = nome;
  }
  public Pessoa(String nome, String cpf) {
    this.nome = nome;
    this.cpf = cpf;
  }
  public Peesoa(String nome, String cpf, String
      endereco) {
    this.nome = nome;
    this.cpf = cpf;
    this.end = endereco;
  } }
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Invocando um construtor de outro construtor

```
public class Pessoa {
        //definição dos atributos e métodos
        public Pessoa(String nome) {
                this.nome = nome;
        }

        public Pessoa(String nome, String cpf) {
                this(nome);
                this.cpf = cpf;
        }
        public Pessoa(String nome, String cpf, String
            endereco) {
                this(nome, cpf);
                this.endereco = endereco;
        }
}
```

Invocando um construtor de outro construtor

Importante!

Quando um construtor é invocado de outro construtor, tal comando deve ser o **primeiro** do bloco

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS