Tecnologia em Análise e Desenvolvimento de Sistemas

Programação Orientada a Objetos

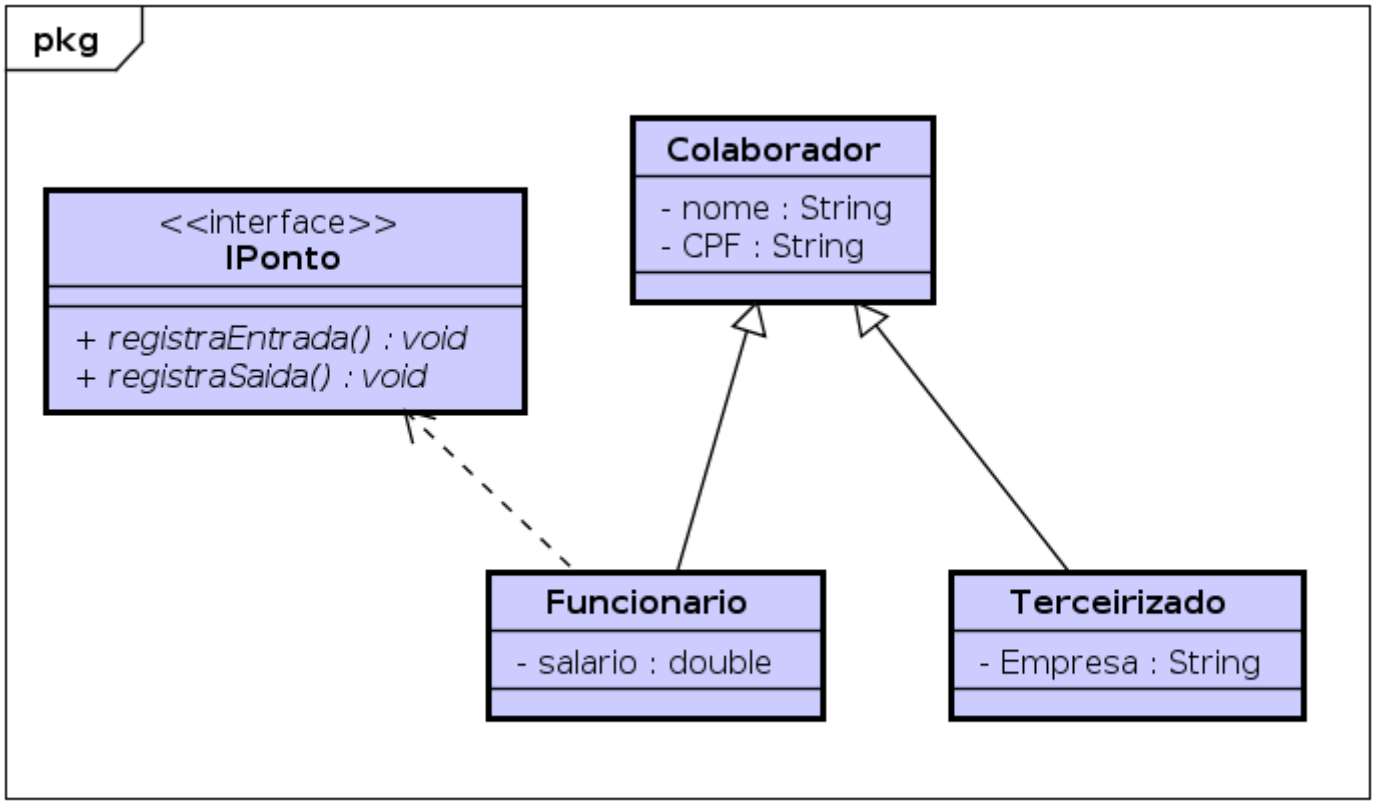
Relacionamentos entre objetos

1º/2018

Relembrando ...

- Qual o principal tipo de relacionamento entre classes estudado até o momento?
  - **Herança**: relacionamento entre uma subclasse e uma superclasse na qual o subclasse possui todas as características e comportamentos disponibilizadas na superclasse.
    - A partir da herança é possível estudar o conceito de polimorfismo (os dois!!!)

Herança


pkg
Colaborador
- nome : String
- CPF : String
<<interface>>
IPonto
+ registraEntrada() : void
+ registraSaida() : void
Funcionario
- salario : double
Terceirizado
- Empresa : String
powered by Astah

```java
package heranca.model;
public abstract class Colaborador {
  private String nome;
  private String CPF;
  public Colaborador(String nome, String CPF) {
    this.nome = nome;
    this.CPF = CPF;
  }
  public String getNome() {
    return nome;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
  public String getCPF() {
    return CPF;
  }
  public void setCPF(String CPF) {
    this.CPF = CPF;
  }
}
```

```java
package heranca.model;
public class Terceirizado extends Colaborador {
  private String empresa;
  public Terceirizado(String nome, String CPF, String empresa) {
    super(nome, CPF);
    this.empresa = empresa;
  }
  public String getEmpresa() {
    return empresa;
  }
  public void setEmpresa(String empresa) {
    this.empresa = empresa;
  }
}
```

```java
package heranca.model;
public class Funcionario extends Colaborador
     implements Iponto{
  private final int MAX = 1000;
  private String[] registroPonto;
  private int ultimoRegistro;
  public Funcionario(String nome, String CPF) {
    super(nome, CPF);
    this.registroPonto = new String[this.MAX];
    this.ultimoRegistro = -1;
  }
  @Override
  public void registrarEntrada(String agora) {
    agora = "ENTRADA: " + agora;
    registrar(agora);
  }
  @Override
  public void registrarSaida(String agora) {
    agora = "SAÍDA: " + agora;
    registrar(agora);
  }
  private void registrar(String txt){
    if(this.ultimoRegistro < this.MAX){
      this.ultimoRegistro += 1;
      this.registroPonto[this.ultimoRegistro] = txt;
    }
  }
}
```

```java
package heranca.model;
public interface Iponto {
  public void registrarEntrada(String agora);
  public void registrarSaida(String agora);
}
```

Outros relacionamentos

- Na orientação a objetos existem outras formas de relacionar objetos:
  - Agregação;
  - Composição;
  - Associações.

Agregação – todo parte

- A agregação é um relacionamento entre duas classes e que estabelece que uma instância de uma classe agrupa uma ou mais instâncias de outra;
- Dá a ideia de que um objeto, para estar completo, isto é, para estar apto a desempenhar seu papel no programa, deve estar associado a um ou mais objetos;
- A agregação também é chamada e **relacionamento todo/parte**, isso porque um objeto (todo) pode ser formado por outros objetos (partes);

Agregação – todo parte

Nesta representação temos que a ClasseA é o todo e a ClasseB é a parte que compõe a ClasseA

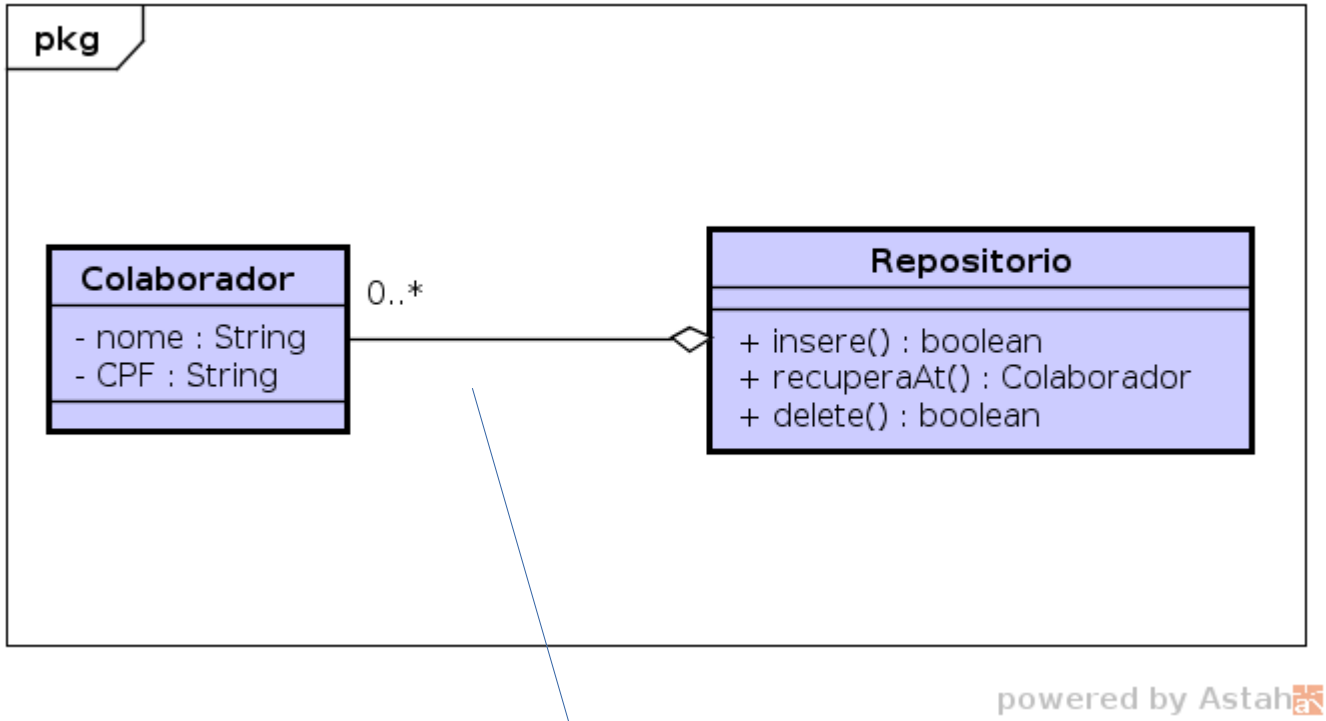
Classe parte

Classe todo

Uma agregação é representada como uma linha que conecta as classes relacionadas, com um diamante (losango) branco perto da classe que representa o todo.

Exemplo15!!!

Agregação – todo parte (exemplo)



Temos que **Um** Repositorio é composto de **0 ou vários** objetos da classe Colaborador.

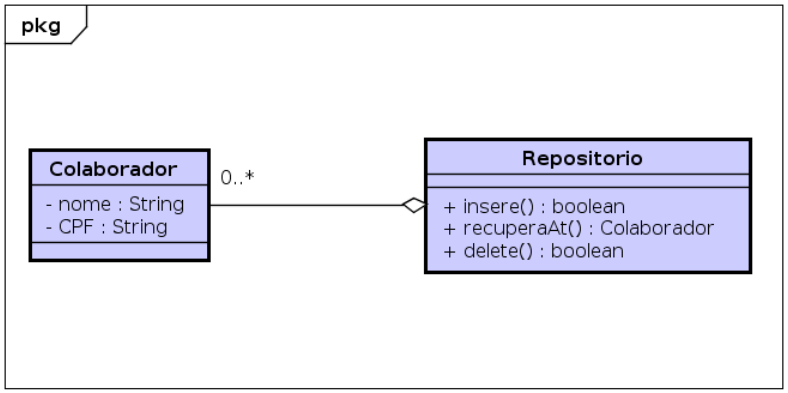
Podemos dizer que vários Colaboradores fazem parte do Repositorio.

Agregação – todo parte (exemplo)

```
package Agrecacao.model;
import heranca.model.Colaborador;
public class Repositorio {
  private final int MAXIMO = 100;
  private Colaborador[] colaboradores;
  private int cadastrados;
  public Repositorio() {
    this.cadastrados = 0;
    this.colaboradores = new Colaborador[this.MAXIMO];
  }
  public boolean insere(Colaborador c){
    boolean deuCerto = false;
    if(this.cadastrados < this.MAXIMO){
      this.colaboradores[this.cadastrados++] = c;
      deuCerto = true;
    }
    return deuCerto;
  }
  public Colaborador recuperaAt(int position){
    return this.colaboradores[position];
  }
  public boolean delete(int position){
    this.colaboradores[position] = null;
    return true;
  }
}
```

Agregação – todo parte (exemplo)

```
package Agrecacao.model;
import heranca.model.Colaborador;
public class Repositorio {
  private final int MAXIMO = 100;
  private Colaborador[] colaboradores;
  private int cadastrados;
  public Repositorio() {
    this.cadastrados = 0;
    this.colaboradores = new Colaborador[this.MAXIMO];
  }
  public boolean insere(Colaborador c){
    boolean deuCerto = false;
    if(this.cadastrados < this.MAXIMO){
      this.colaboradores[this.cadastrados++] = c;
      deuCerto = true;
    }
    return deuCerto;
  }
  public Colaborador recuperaAt(int position){
    return this.colaboradores[position];
  }
  public boolean delete(int position){
    this.colaboradores[position] = null;
    return true;
  }
}
```



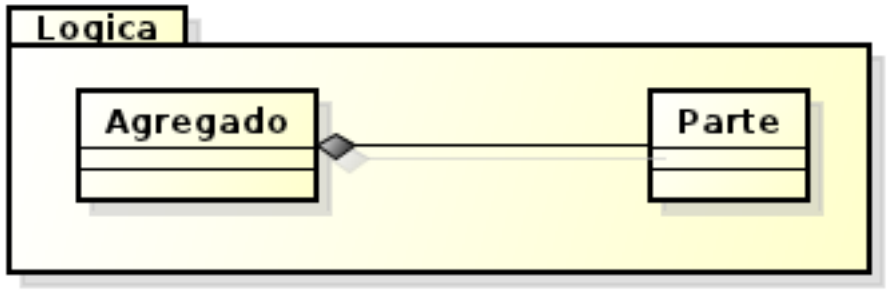
Observe que o Repositório pode ter zero ou vários (definido um máximo) de colaboradores.

Composição – todo parte

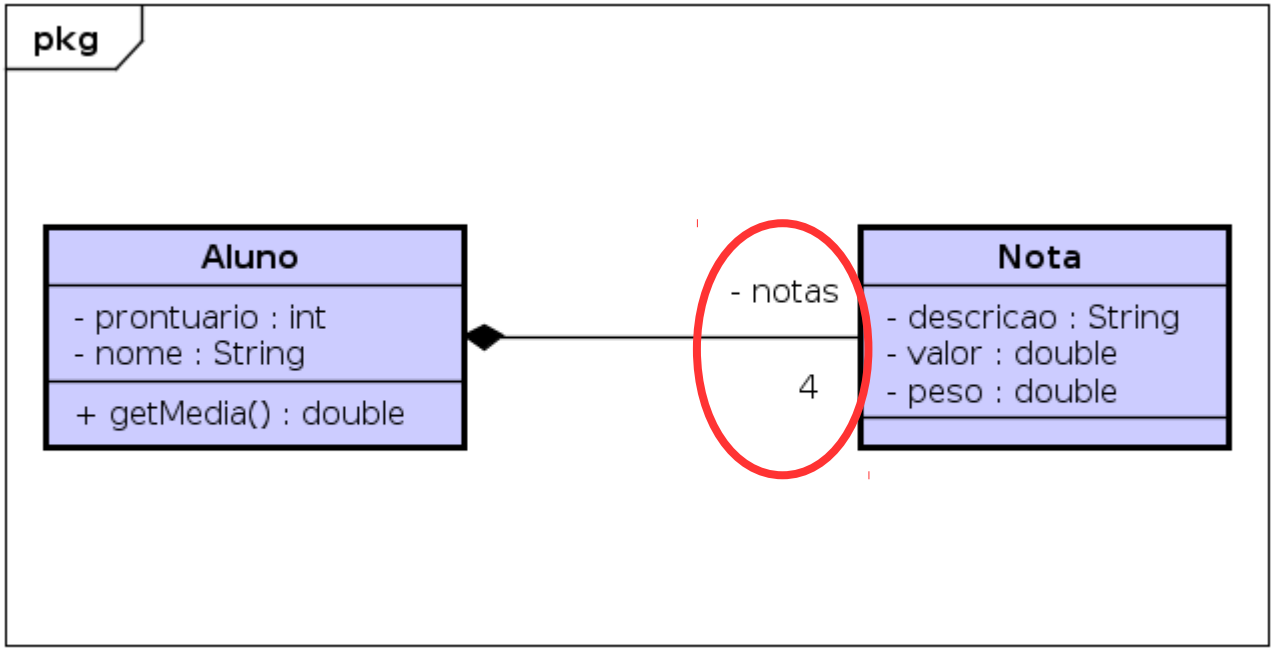
- **Composição**: Um tipo de relação de agregação, também chamada **agregação forte**, em que há um conjunto de requisitos na ligação entre parte e agregado:
  - Uma instância da parte é agregada por uma única instância do agregado (não há compartilhamento da parte);
  - A existência da parte depende da existência do agregado. Assim, a instanciação do agregado precede a instanciação da parte e a destruição do agregado implica na destruição da parte.

Composição – todo parte

- A representação da composição em UML consiste de uma linha com um diamante (losango) preenchido numa das extremidades, tocando a classe agregado.



Composição – todo parte (exemplo)



Composição – todo parte (exemplo)

```
package composicao.model;
public class Nota {
  private String Descricao;
  private double valor;
  private double peso;
  public Nota(String descricao, double valor, double peso) {
    Descricao = descricao;
    this.valor = valor;
    this.peso = peso;
  }
  public String getDescricao() {
    return Descricao;
  }
  public void setDescricao(String descricao) {
    Descricao = descricao;
  }
  public double getValor() {
    return valor;
  }
  public void setValor(double valor) {
    this.valor = valor;
  }
  public double getPeso() {
    return peso;
  }
  public void setPeso(double peso) {
    this.peso = peso;
  }
}
```

Composição – todo parte (exemplo)

```java
package composicao.model;
public class Aluno {
  private int prontuario;
  private String nome;
  private Nota[] notas;

  public Aluno(int prontuario, String nome,
     Nota nota1, Nota nota2, Nota nota3,
     Nota nota4) {
    this.prontuario = prontuario;
    this.nome = nome;
    this.notas = new Nota[4];
    this.notas[0] = nota1;
    this.notas[1] = nota2;
    this.notas[2] = nota3;
    this.notas[3] = nota4;
  }

  public Aluno(int prontuario, String nome) {
    this.prontuario = prontuario;
    this.nome = nome;
    this.notas = new Nota[4];
    for(int i=0; i<4; i++)
      this.notas[i] = null;
  }
```

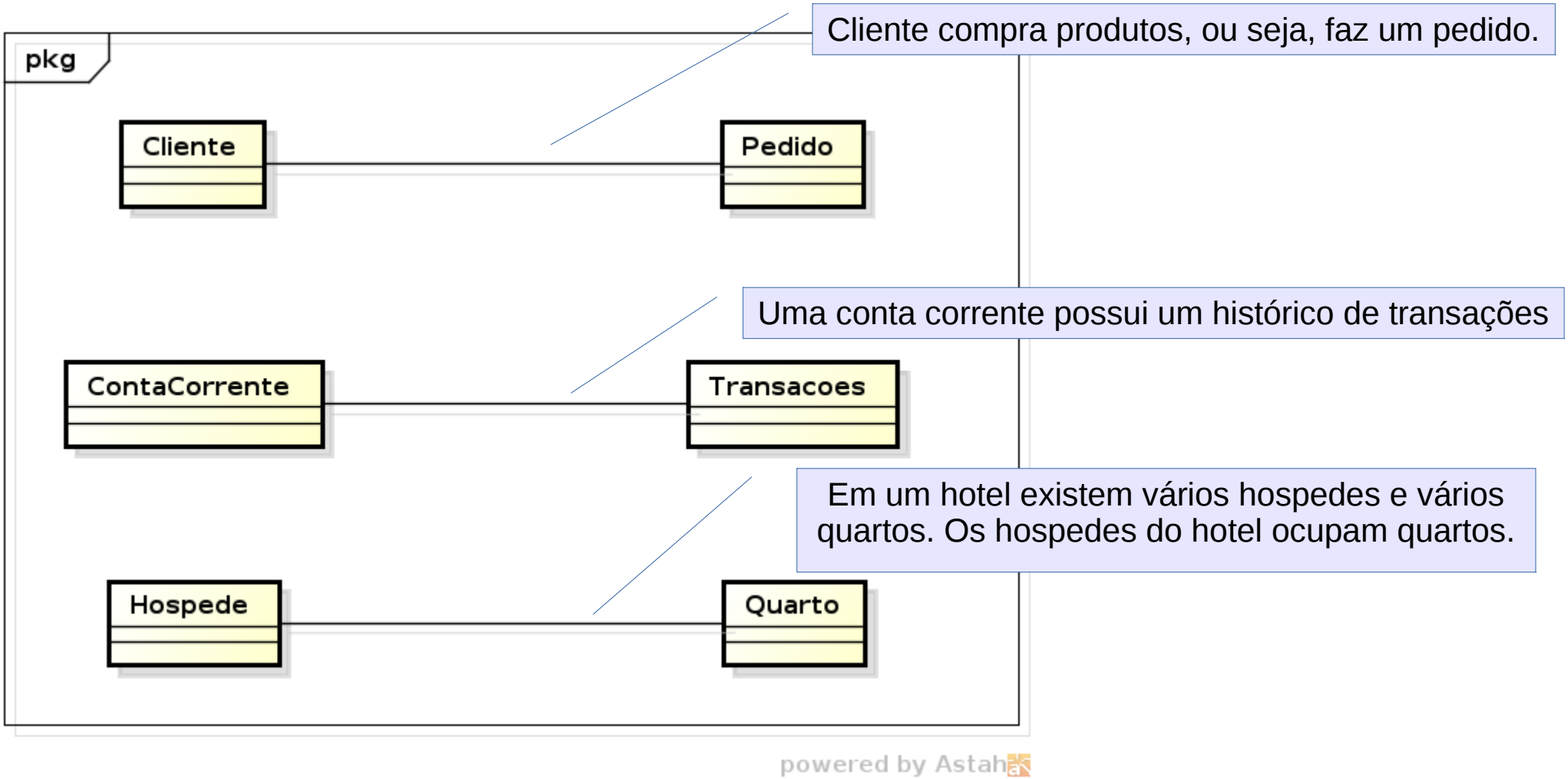
```java
  public double getMedia(){
    double media;
    double somaPeso;
    media = 0;
    somaPeso = 0;
    for(int i=0; i<4; i++){
      if(this.notas[i] == null){
        media = 0;
        i = 5;
      }else{
        media += this.notas[i].getValor() *
           this.notas[i].getPeso();
        somaPeso += this.notas[i].getPeso();
      }
    }
    if(media > 0 && somaPeso != 0){
      media = media / somaPeso;
    }
    return media;
  }
  public int getProntuario() {
    return prontuario;
  }
  public void setProntuario(int prontuario) {
    this.prontuario = prontuario;
  }
  public String getNome() {
    return nome;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

Associações

- Um ponto importante a ser entendido em um sistema orientado a objetos é o fato de que objetos podem se relacionar uns com os outros;

- A existência de um relacionamento entre dois objetos possibilita a troca de mensagens entre os mesmos, desta forma, o relacionamento entre objetos permite que eles colaborem entre si a fim de produzir as funcionalidades do sistema;

Associações



Multiplicidade

- As associações permitem representar a informação dos limites inferior e superior da **quantidade de objetos** aos quais um objeto pode estar associado. Estes limites são denominados **multiplicidade**;

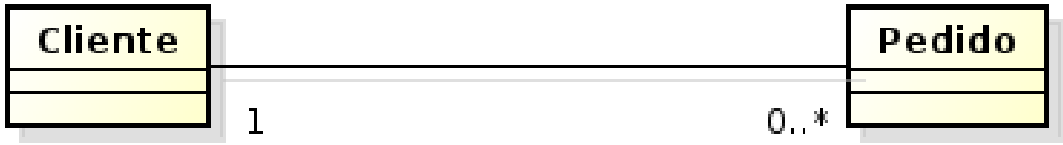
- Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha que a representa;

Multiplicidade

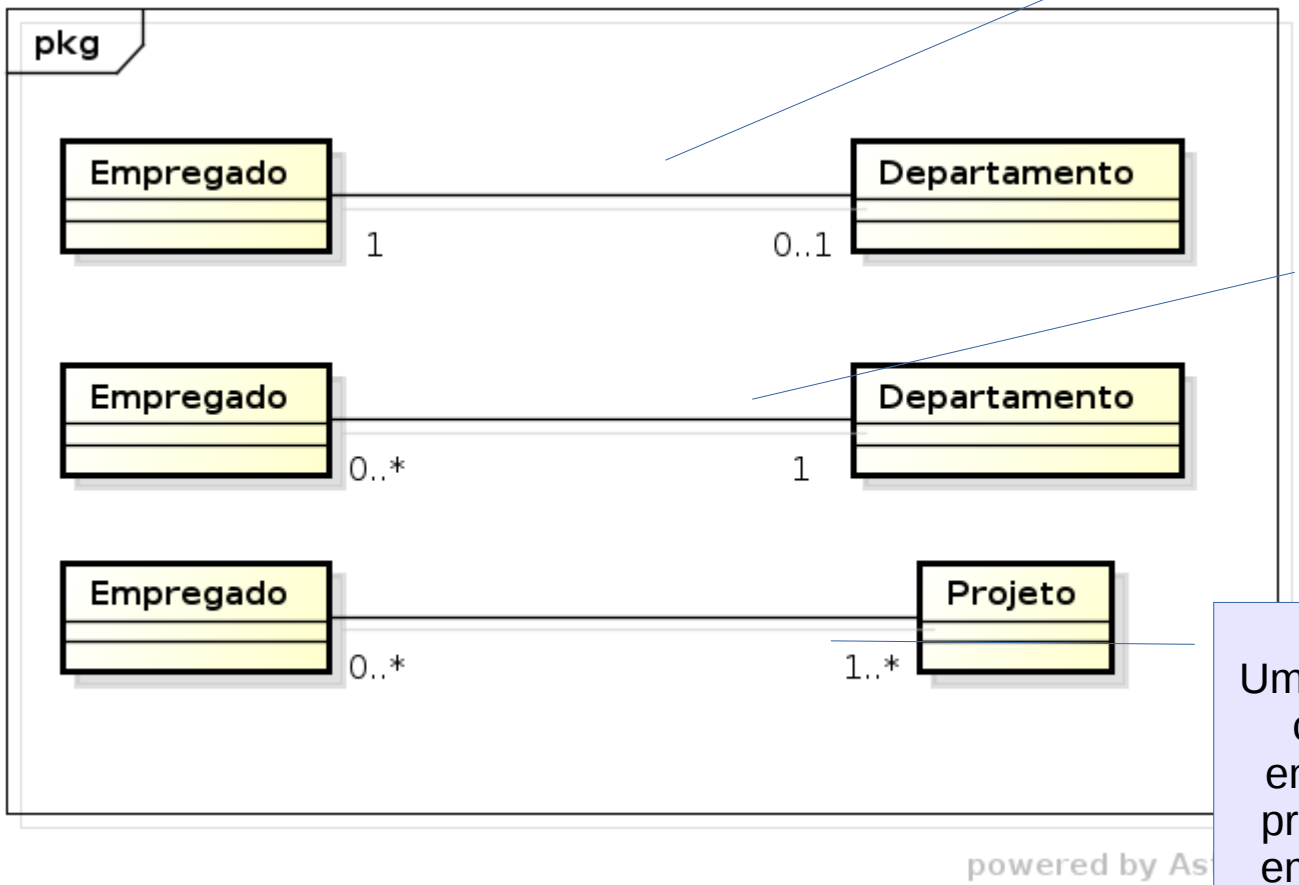
Nome	Simbologia
Apenas um	1
Zero ou muitos	0..*
Um ou muitos	1..*
Zero ou um	0..1
Intervalo específico	n..m

Multiplicidade - exemplo

- Neste exemplo considere duas classes, *Cliente* e *Pedido*, e uma associação entre elas;
- A leitura desta associação nos informa que pode haver um objeto Cliente que esteja associado a vários objetos da classe Pedido;
- Além disso, essa leitura nos informa que pode haver um objeto da classe Cliente que não esteja associado a pedido algum;
- Representa também a informação de que um objeto Pedido está associado a um, e somente um, objeto da classe Cliente.



Outros exemplos



Conectividade Um para Um
Indica que um empregado pode gerenciar um e somente um departamento. Por sua vez, o departamento possui um único gerente. Observe que nem todo empregado é gerente.

Conectividade Um para Muitos
Um empregado está lotado em um único departamento, mas um departamento pode ter diversos empregados. Porém o departamento não precisa ter empregado algum.

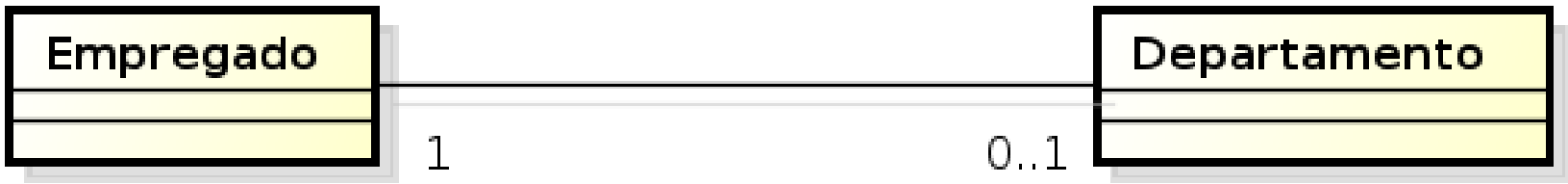
Conectividade Muitos para Muitos
Um projeto pode ter diversos empregados como trabalhadores. Além disso, um empregado pode trabalhar em diversos projetos. Um empregado deve trabalhar em pelo menos um projeto e um projeto pode não ter empregados.

Participação

- Uma característica importante de uma associação está relacionada à necessidade ou não da existência dessa associação entre objetos. Essa característica é denominada **participação**;

- A participação pode ser *obrigatória* ou *opcional*. Se o valor mínimo da multiplicidade de uma associação é igual a 1 (um), significa que a participação é obrigatória, caso contrário é opcional;

Participação



- A multiplicidade de valor 1 próxima a Empregado indica que um objeto da classe Departamento só pode existir se estiver associado a um objeto Empregado. **Não pode-se instanciar um departamento sem um empregado (no caso o gerente);**
- Para objetos da classe Departamento a participação é obrigatória. No entanto, para objetos de Empregado, essa mesma associação é parcial (pode haver empregados que não estejam associados a um departamento);

Nome da associação e direção de leitura

- Com o objetivo de deixar a associação mais clara, a UML define alguns recursos de notação, entre eles o nome da associação e a direção da leitura;
- O nome da associação é posicionado na linha da associação, entre as classes envolvidas;
- A direção de leitura indica como a associação deve ser lida. Essa direção é representada por um pequeno triângulo posicionado próximo a um dos lados do nome da associação;
- O nome de uma associação deve fornecer algum significado semântico à mesma;

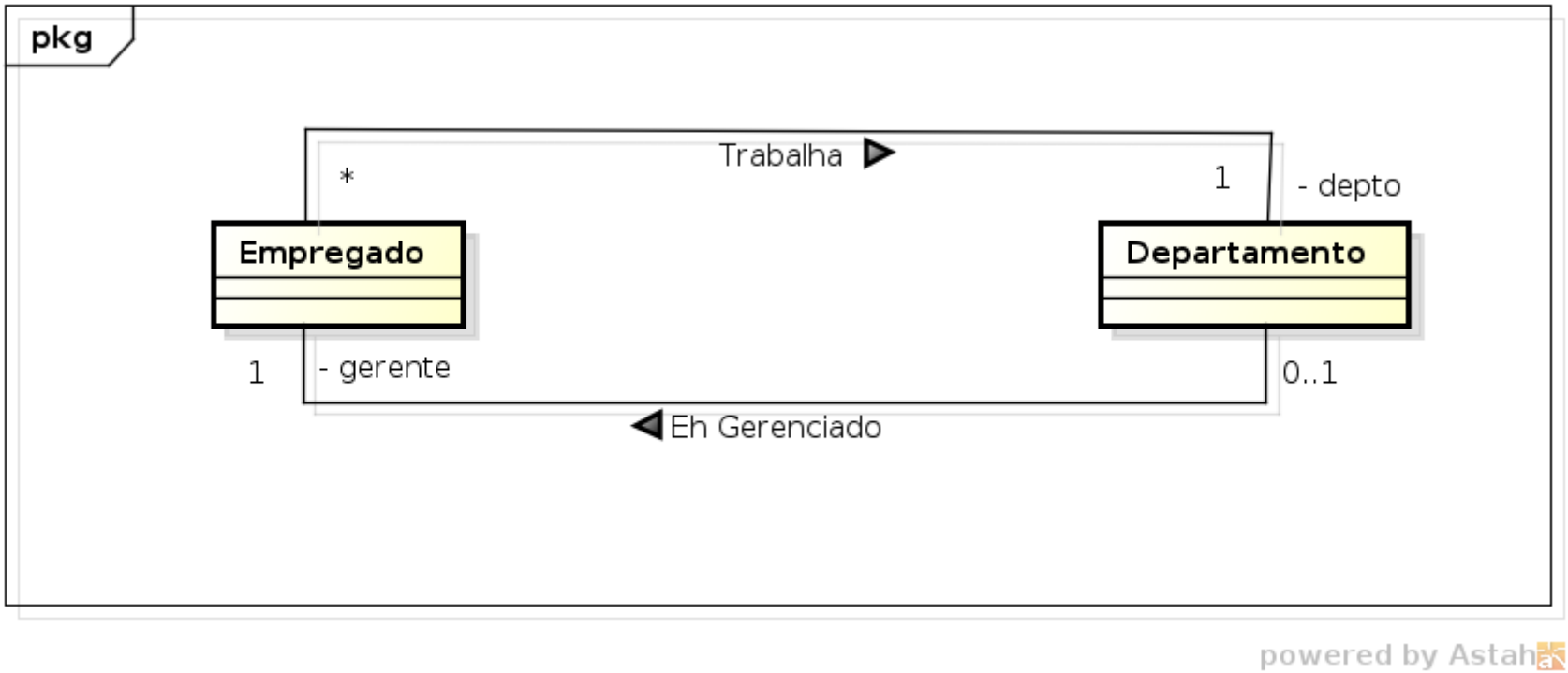
Nome da associação e direção de leitura

- Neste exemplo, temos que uma Organização contrata vários Indivíduos. A direção de leitura indica que a Organização é o objeto que executa a ação de contratar e não o contrário.

Papéis

- Outra notação definida pela UML para aumentar o significado de uma associação é o papel. Isso é o papel que cada objeto da classe irá representar na associação;
- Quando um objeto participa de uma associação, ele tem um papel (*role*) específico nela;

Papéis - exemplo



Trabalhando

- **Exercício de Fixação**
  - 11 e 12