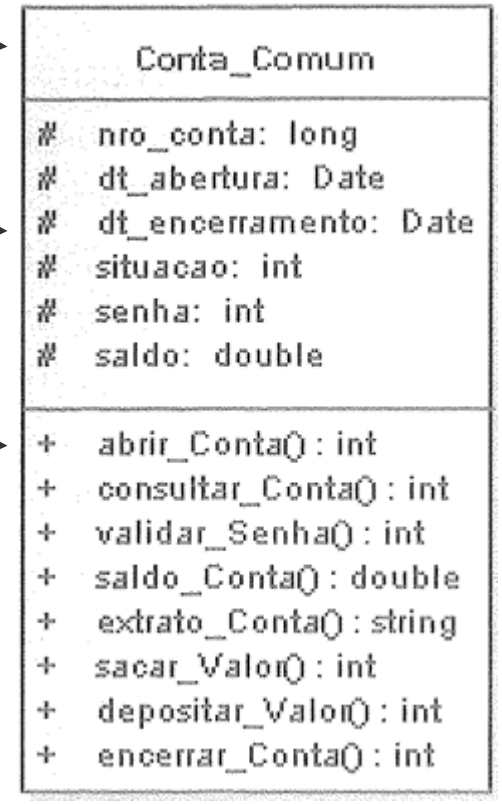
Diagrama de Classes

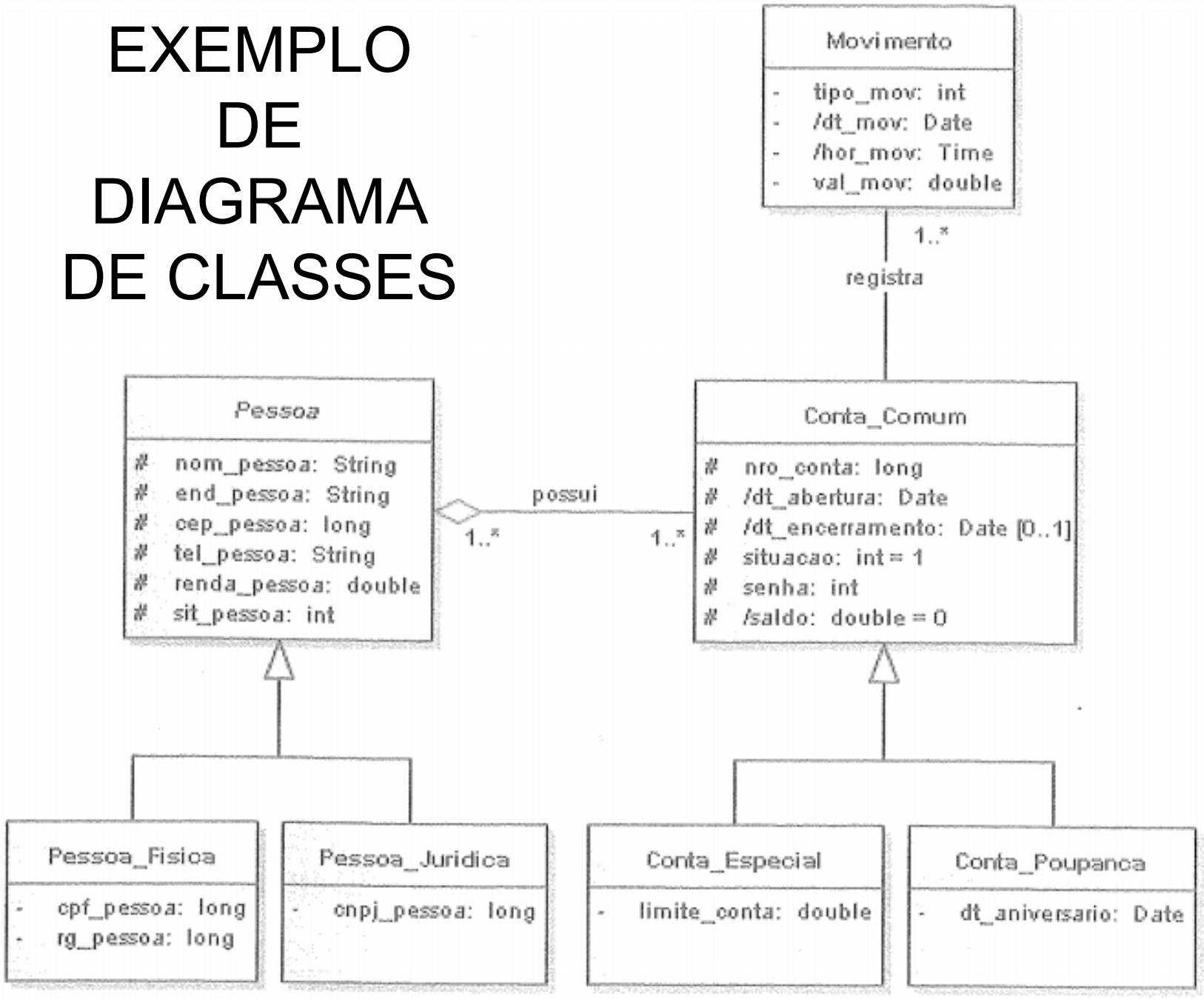
- Seu principal objetivo é permitir a visualização das classes que vão compor o sistema, seus atributos, métodos e como essas classes se relacionam entre si
- O Processo Unificado recomenda a utilização deste diagrama ainda na fase de Análise: nesta fase ainda não são representados os métodos, que são identificados na fase de Projeto, nos diagramas de interação (Seqüência)

Diagrama de Classes

- A representação de uma classe é feita assim:
  - Nome da Classe
  - Atributos
  - Métodos



EXEMPLO DE DIAGRAMA DE CLASSES



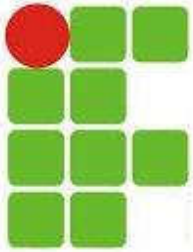


Diagrama de Classes

- Visibilidade no Diagrama de Classes:
  - (-) Privado: Somente a classe pode ver o valor do atributo
  - (+) Público: Qualquer classe pode ver o valor do atributo
  - (~) Pacote: Somente as classes do mesmo pacote podem ver o valor do atributo
  - (#) Protegido: Somente a própria classe ou suas filhas por herança podem ver o valor do atributo

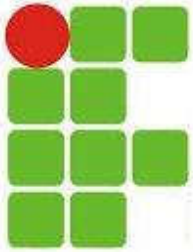


Diagrama de Classes

- Métodos podem retornar um valor de um tipo e também podem receber valores como parâmetros, cada um com um tipo
- O nome + tipo de retorno + parâmetros é a assinatura do método
- Lembrando que para fazer override (sobrescrita) ou overload (sobrecarga) é preciso atentar para isso!

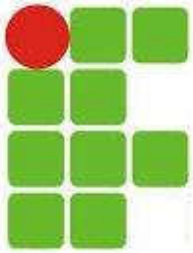
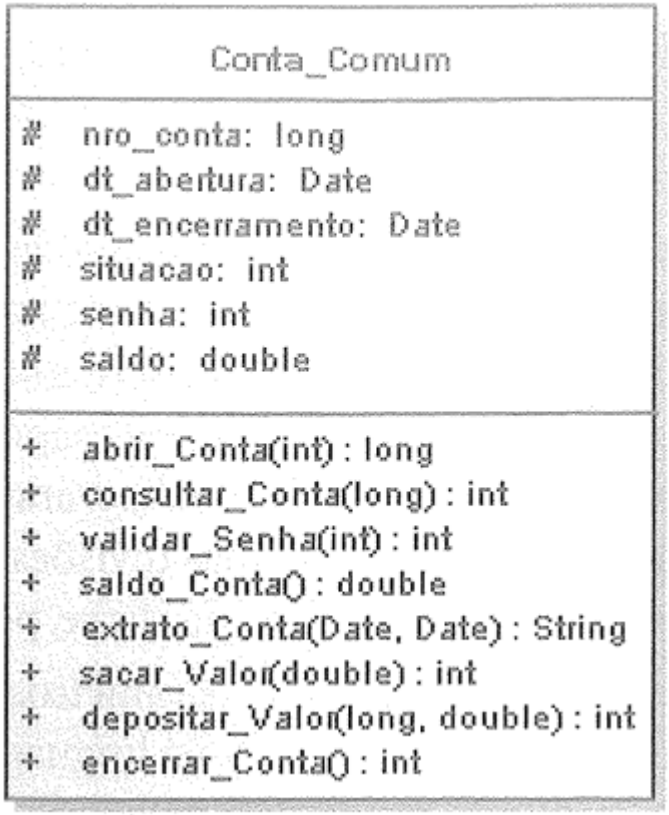


Diagrama de Classes

- Exemplo de representação de métodos em um Diagrama de Classes:
  - Não é preciso colocar o nome do parâmetro, apenas o tipo
  - A visibilidade também aparece como nos atributos
  - O tipo de retorno aparece no final, após os dois pontos



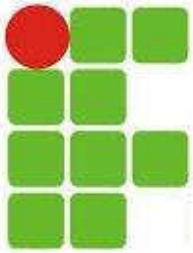


Diagrama de Classes

- Representação dos atributos e métodos dos exemplos anteriores em uma classe Java:

```
public class Conta_Comum {
    protected long nro_conta;
    protected Date dt_abertura;
    protected Date dt_encerramento;
    protected int situacao;
    protected int senha;
    protected double saldo;

    public Conta_Comum(){
    }
    public void finalize() throws Throwable {
    }
    public long abrir_Conta(int senha){
        return 0;
    }
    public int consultar_Conta(long nro_conta){
        return 0;
    }
```

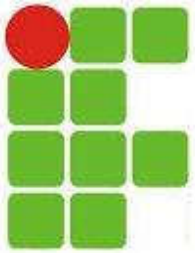


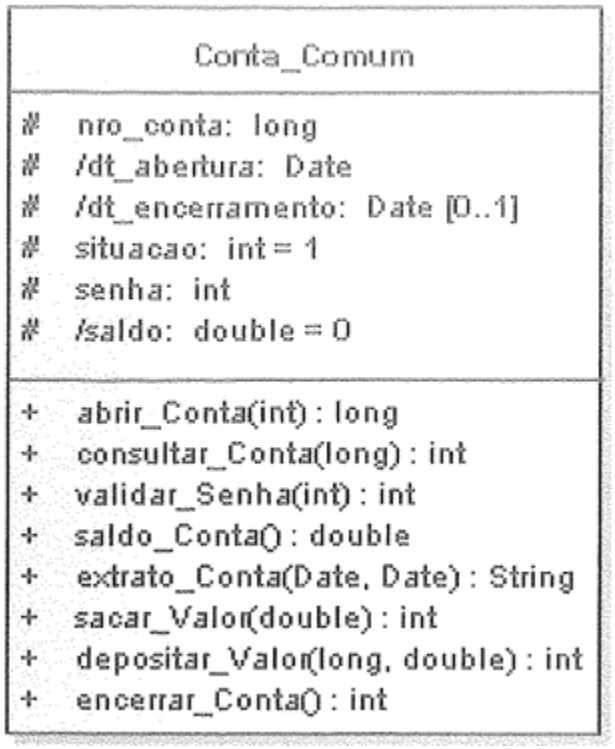
Diagrama de Classes

- Continuação:

```
    public int validar_Senha(int senha){
        return 0;
    }
    public double saldo_Conta(){
        return 0;
    }
    public String extrato_Conta(){
        return "";
    }
    public int sacar_Valor(double valor){
        return 0;
    }
    public int depositar_Valor(long nro_conta, double valor){
        return 0;
    }
    public int encerrar_Conta(){
        return 0;
    }
}
```

Diagrama de Classes

- Adicionando Detalhes:
  - A / antes dos atributos dt_abertura e dt_encerramento significa que os valores desses atributos sofrem algum tipo de cálculo
  - A multiplicidade [0..1] depois de dt_encerramento significa que uma conta pode ou não ter essa data



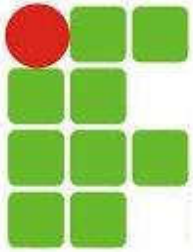
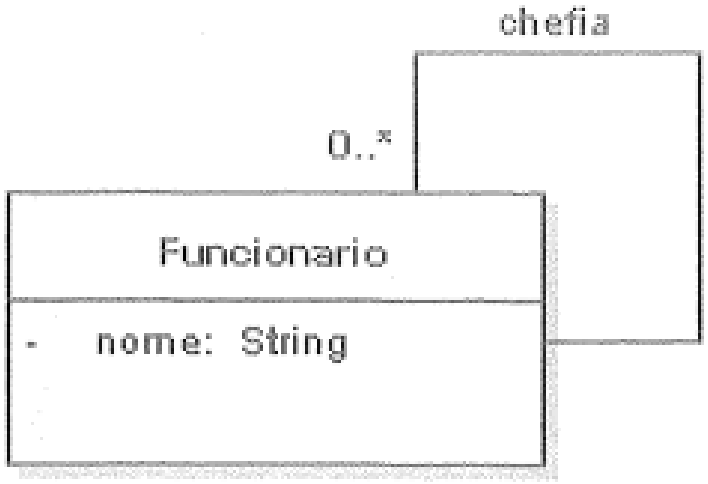


Diagrama de Classes

- Relacionamentos ou Associações:
  - Descrevem um vínculo que permite a troca de informações e colaboração para execução de processos dentro do sistema
  - São representadas por linhas que podem ter nomes informando o tipo de associação
- Podem ser:
  - Unárias, Binárias, Ternárias, Agregação, Composição, Generalização/Especialização, Classes Associativas, Realização ou Dependência

Associação Unária (ou Reflexiva)

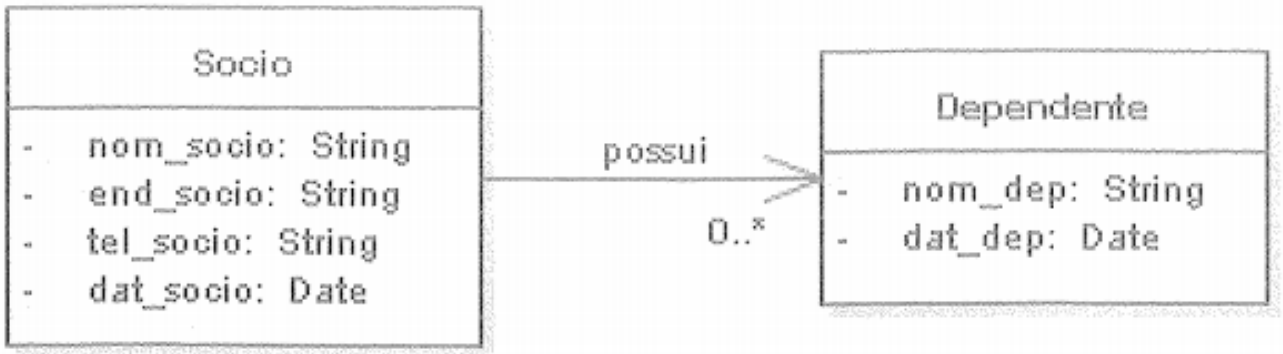
- Ocorre quando existe um relacionamento de um objeto da classe com objetos da mesma classe:



- Um funcionário pode ser chefe de nenhum ou vários outros funcionários

Associação Binária

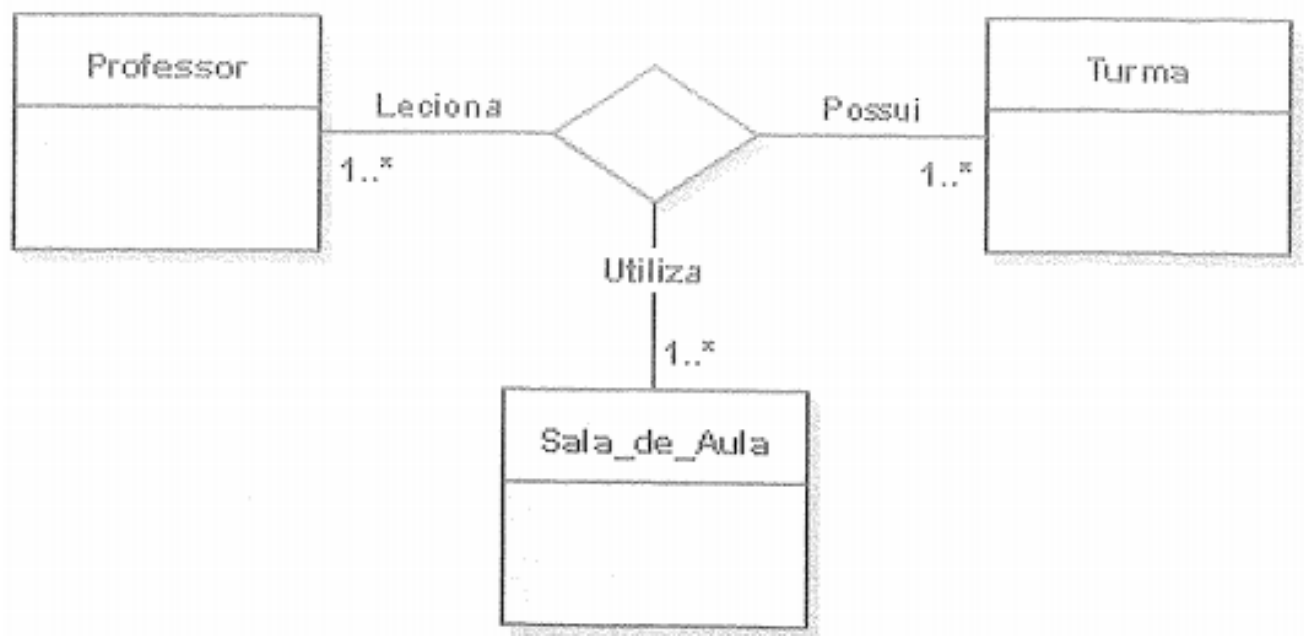
- É um relacionamento entre objetos de duas classes distintas:



- Um sócio pode possuir zero ou vários dependentes
- Um depentente obrigatoriamente estará vinculado a um sócio (implícito = 1..1)

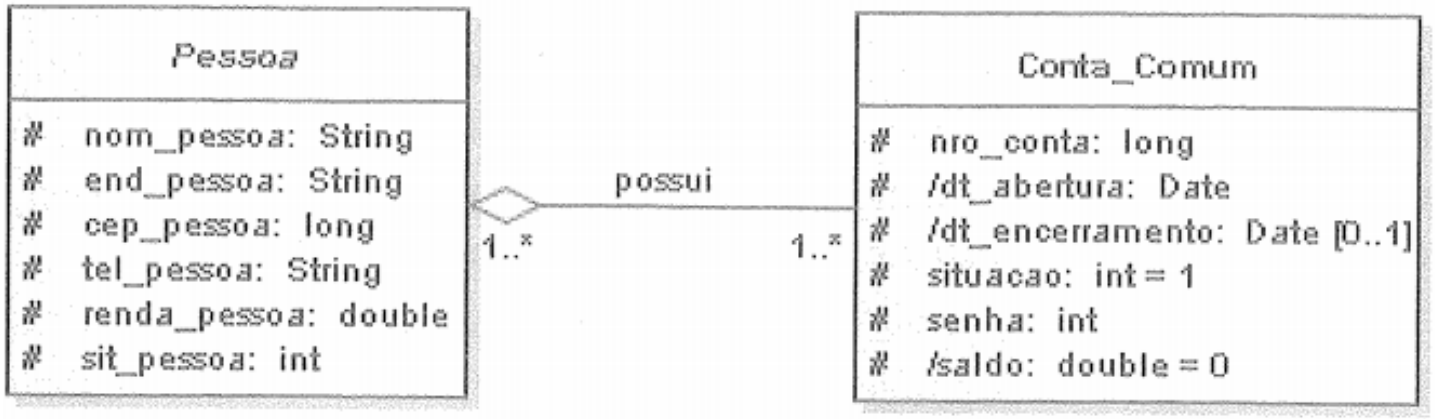
Associação Ternária (ou N-ária)

- São associações que conectam objetos de três o mais classes
- São representadas por um losango que recebe todas as ligações da associação:



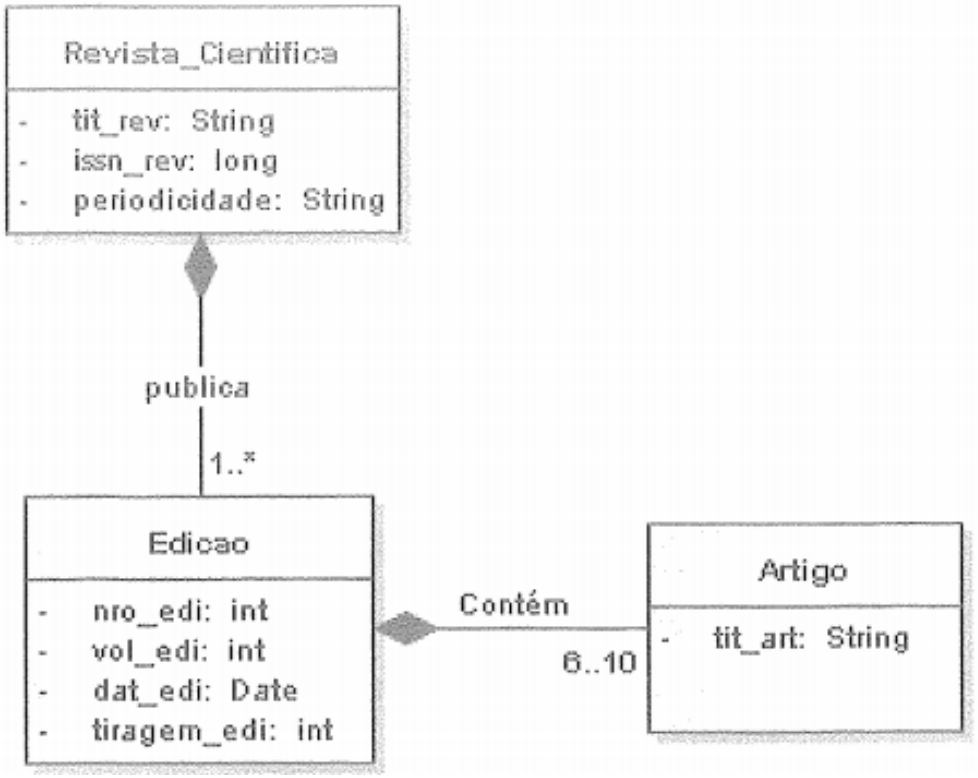
Agregação

- Por vezes as informações de um objeto-todo precisam ser complementadas por informações contidas em outros objetos-parte
- Para isso utiliza-se a Agregação
- É representada por um losango (objeto-todo):



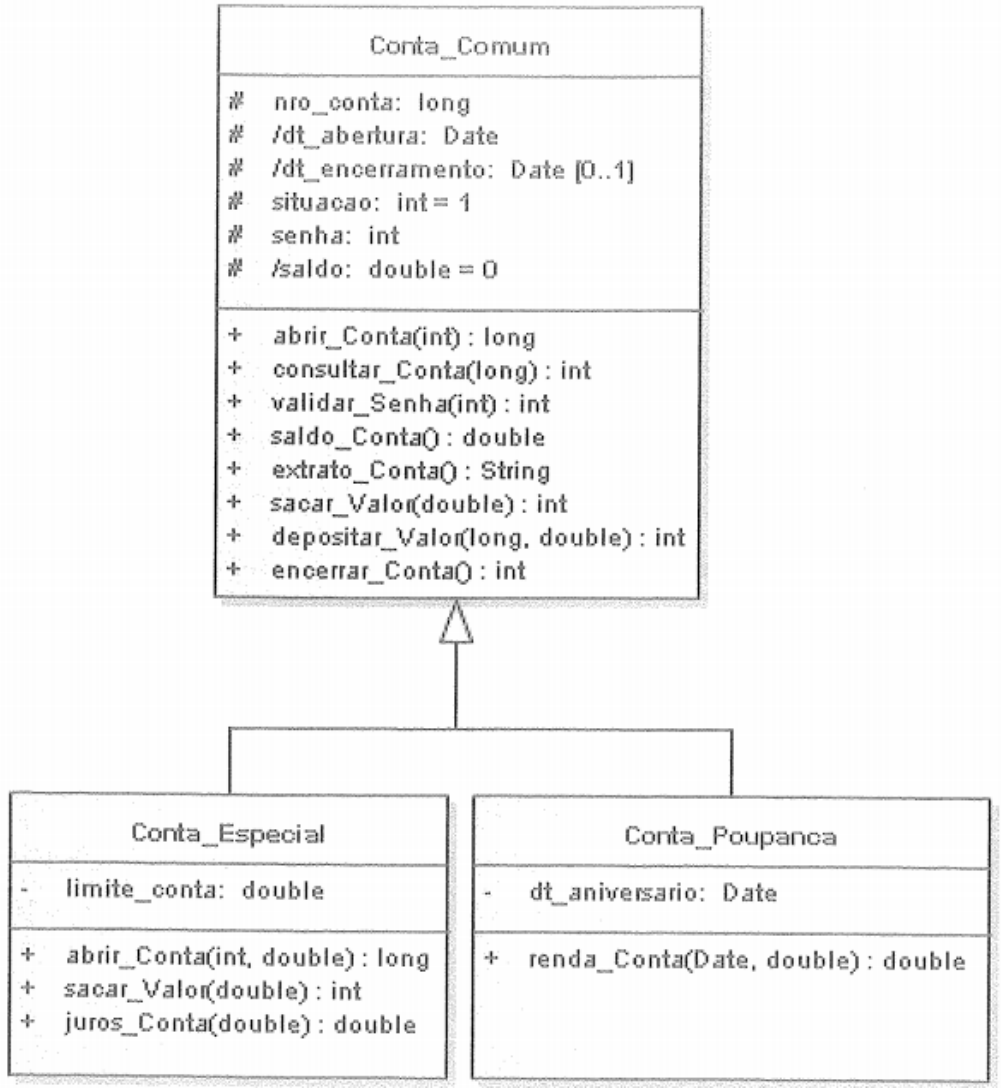
Composição

- É uma associação mais forte que a Agregação onde os objetos-parte precisam estar associados a um **único** objeto-todo e só podem ser destruídos por ele:
  - Nesse caso os artigos só podem ser inéditos, ou seja, estão vinculados a uma única edição!



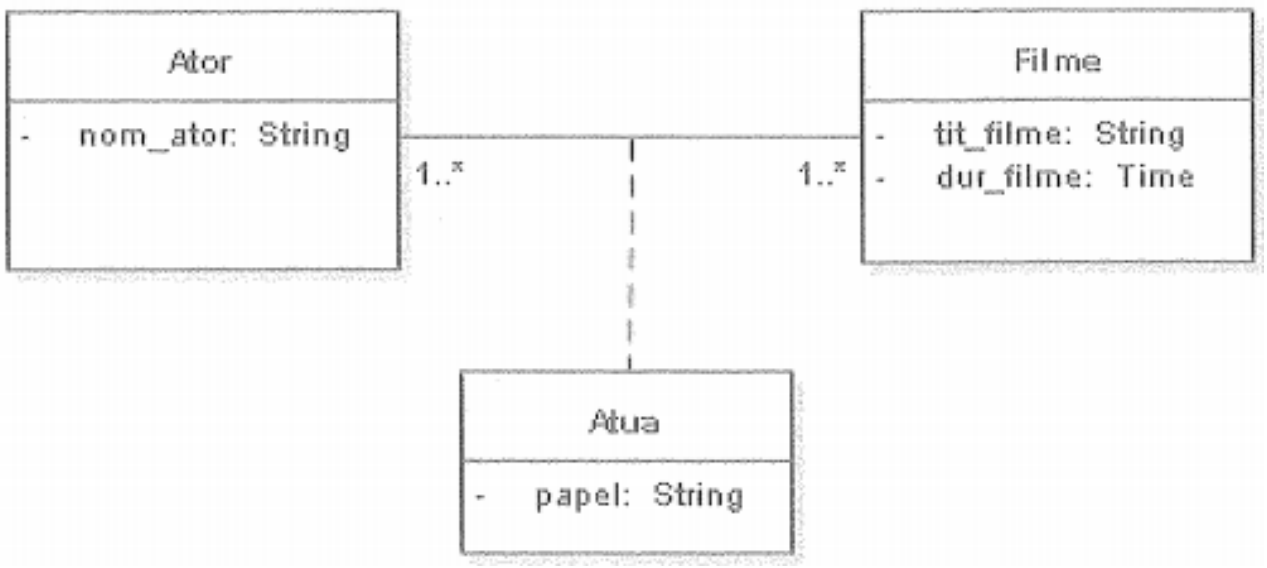
Generalização/Especialização

- Objetiva representar as relações de herança e possível polimorfismo
- Similar ao que foi visto no Diagrama de Casos de Uso



Classe Associativa

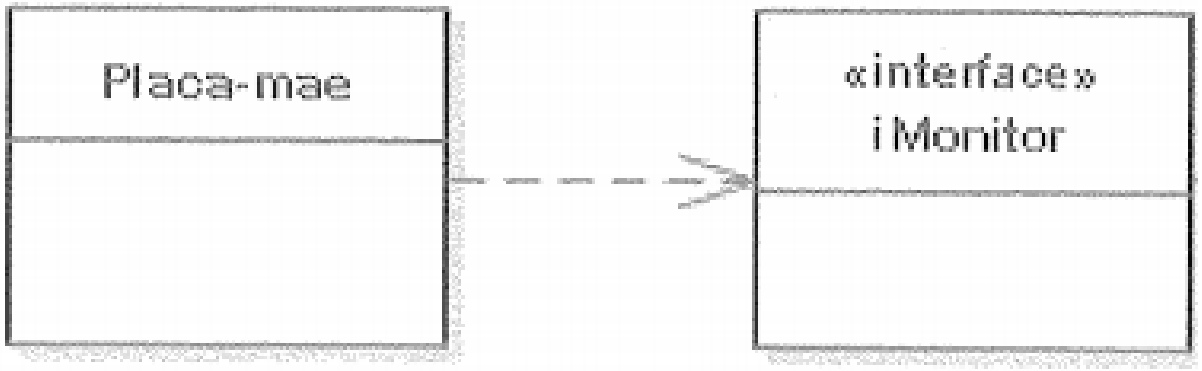
- Ocorrem em associações com multiplicidade ‘muitos’ em ambos os lados



- Nesse caso há também uma informação importante na associação em si: o **papel**

Dependência

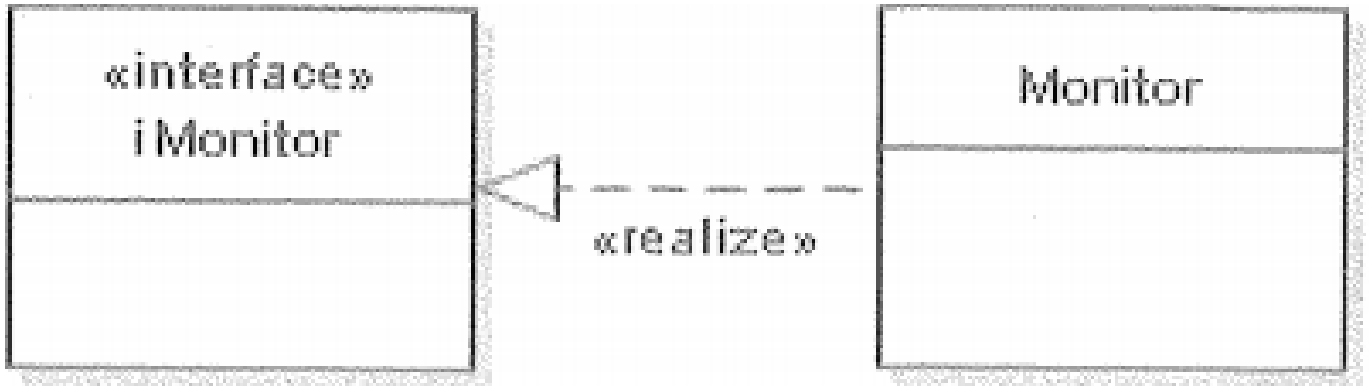
- Representa o grau de dependência de uma classe em relação a outra



- Nesse caso a placa-mãe depende da interface iMonitor, provavelmente para seguir um padrão da indústria, pois interfaces possuem só as assinaturas dos métodos

Realização

- Esta associação é utilizada para identificar classes responsáveis por executar funções para outras classes
- Nesse tipo de relacionamento se herda o comportamento de uma classe mas não sua estrutura (seria o 'implements' do Java)

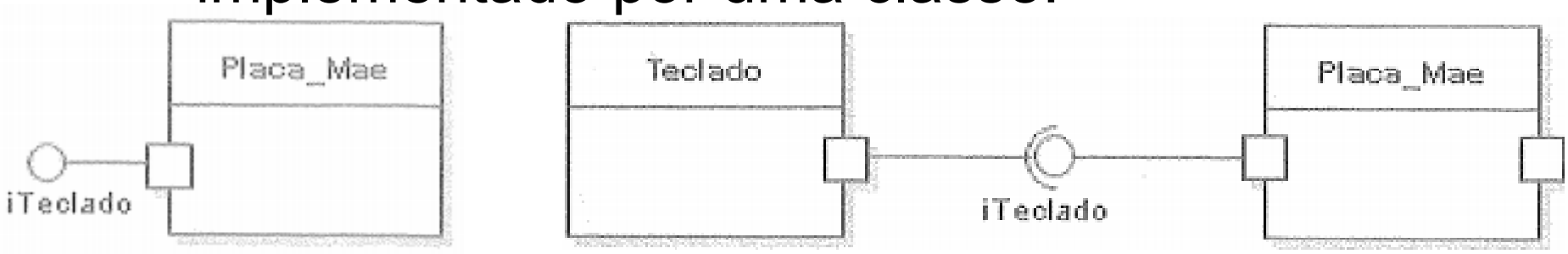


Portas

- Representam um ponto de comunicação entre um classificador e o ambiente
- São representadas por pequenos quadrados nas bordas da classe
- São mais utilizadas nos diagramas de estrutura composta e de componentes

Interfaces

- Podem ser fornecidas ou requeridas:
  - Fornecidas: descrevem um serviço implementado por uma classe:



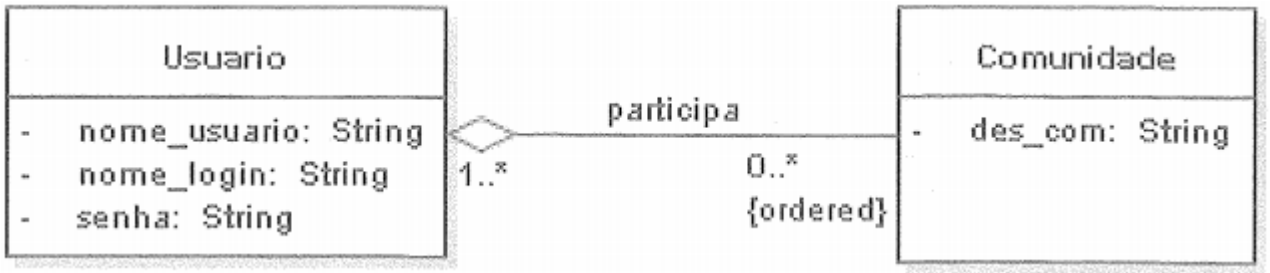
  - Requeridas: descrevem um serviço que outras classes devem fornecer à classe em questão:

Restrições

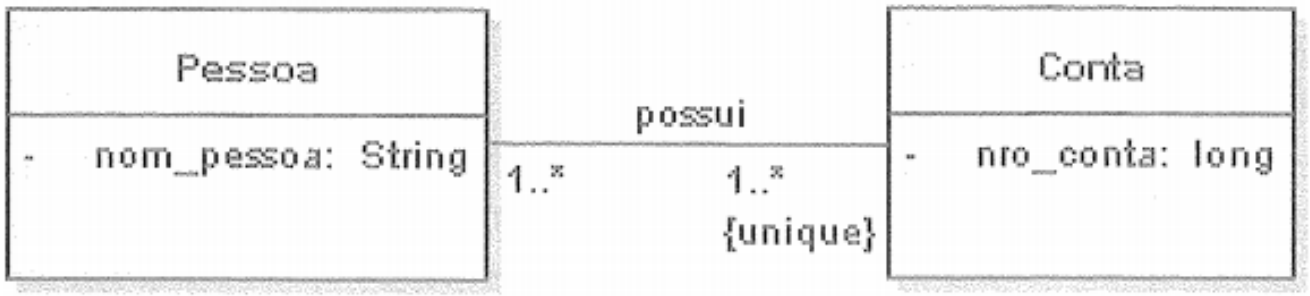
- São condições a serem validadas durante a implementação dos métodos de uma classe, suas associações ou atributos
- São representadas por textos entre chaves
- Podem ser utilizadas para detalhar requisitos não-funcionais

Restrições

- Podem indicar também a forma de organização de alguns dados:



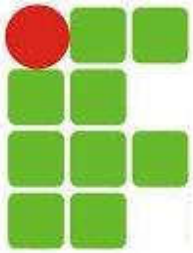
- Também podem indicar outras restrições à coleções (unique, bag, sequence):



Atributos Static

- São atributos cujo valor é o mesmo para todos os objetos de uma determinada classe
- São represe 

a sublinhada

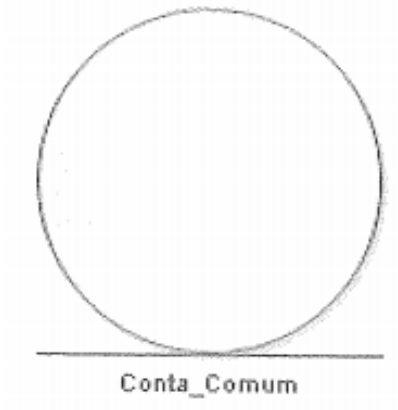


Estereótipos

- São representações que permitem destacar alguns componentes no diagrama que têm uma função especial
- É possível a criação de outros estereótipos
- Os principais são:
  - <<entity>>
  - <<boundary>>
  - <<control>>

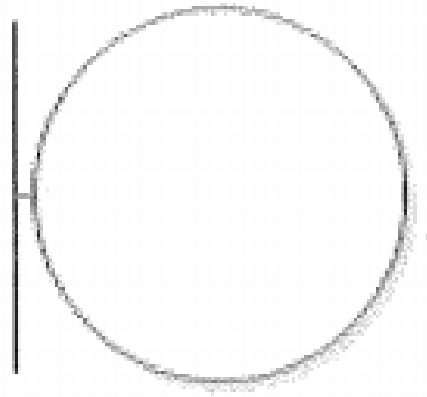
<<entity>>

- Explicitam que uma classe é persistente, ou seja, tem informações que serão armazenadas pelo sistema, provavelmente em um Banco de Dados <<persistente>>, porém podem ser transientes também
- São classes relacionadas com o contexto do software
- Representação:



<<boundary>>

- Representa uma classe que tem comunicação com o meio externo, um ator
- Muitas vezes é associada à própria interface do sistema
- Desnecessária em sistemas muito simples
- Sua representação:



<<control>>

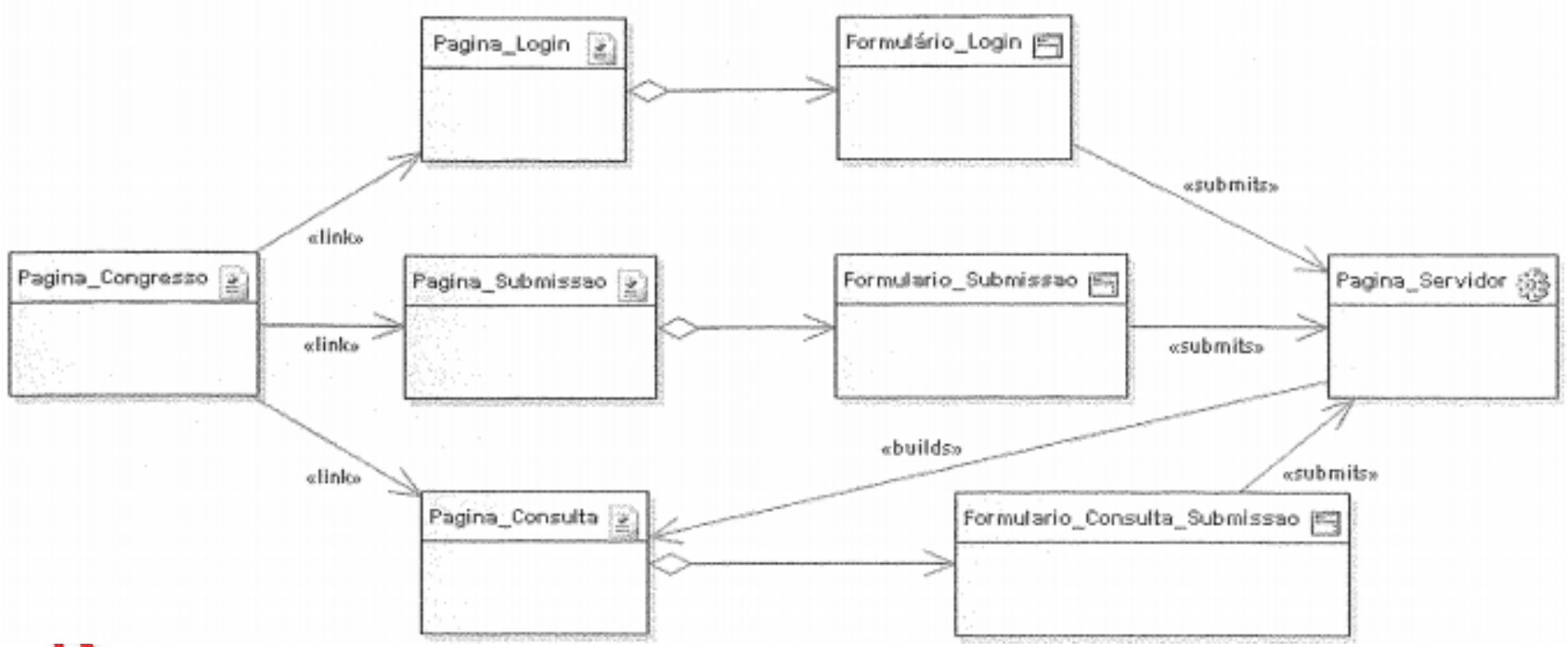
- Identifica as classes que servem de intermediárias entre as classes <<boundary>> e as demais classes do sistema
- Interpretam os eventos ocorridos sobre os objetos <<boundary>>
- Sua representação:

Projeto Navigacional

- Alguns estereótipos podem ser utilizados para identificar certas telas possibilitando a representação da navegabilidade do sistema:
  - <<server page>>
  - <<client page>>
  - <<form>>

Projeto Navigacional

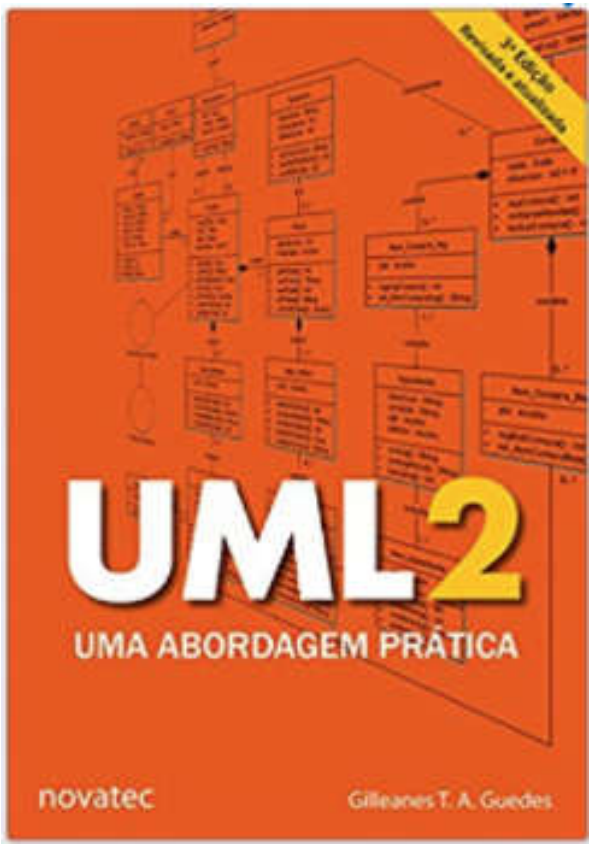
- Exemplo de projeto navegacional:

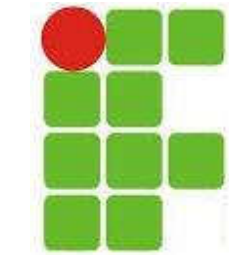


Observe que há outros estereótipos!

Referências

- UML2: Uma Abordagem Prática
  3ª Ed. 2018
  Gilleanes T. A. Guedes





Perguntas?