Unitri
CENTRO UNIVERSITÁRIO DO TRIÂNGULO

Especialização em Desenvolvimento Java
UML e Padrões de Projetos

Prof. Vinícius de Paula

https://github.com/viniciusdepaula/aulas-uml-padroes

Diagrama de Classes

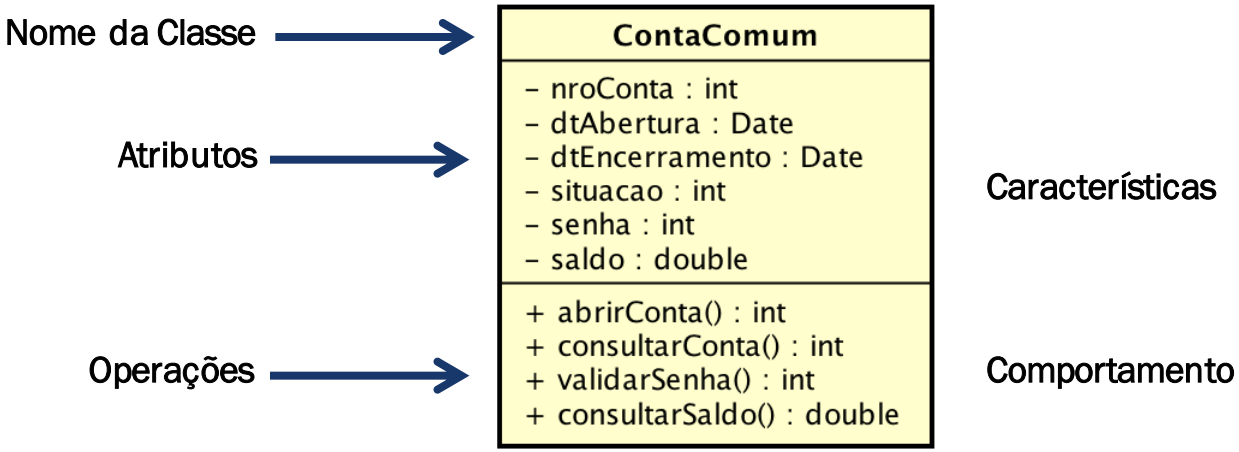
- Seu principal enfoque está em:
  - Permitir a visualização das classes que irão compor o sistema com seus respectivos atributos e métodos;
  - Demonstrar como as classes do diagrama se relacionam, complementam e transmitem informações entre si.
- Apresenta uma **visão estática** de como as classes estão organizadas, preocupando-se em como definir a **estrutura lógica** das mesmas.

Diagrama de Classes

- Na fase de análise, um **modelo conceitual** do diagrama de classes é produzido.
  - As informações que o software necessitará, em termos de classes e seus atributos, bem como as associações entre as classes devem ser representadas
- Já na fase de projeto, com base no modelo conceitual do diagrama de classe é produzido o **modelo de domínio**.
  - A solução do problema é enfocada.
    - Os métodos que as classes poderão conter fazem parte de como o software será desenvolvido e por este motivo eles devem aparecer no modelo de domínio.

Atributos e Métodos

- Classes costumam ter:
  - Atributos que armazenam os dados dos objetos da classe e;
  - Métodos que são funções que uma instância da classe pode executar.



Visibilidade

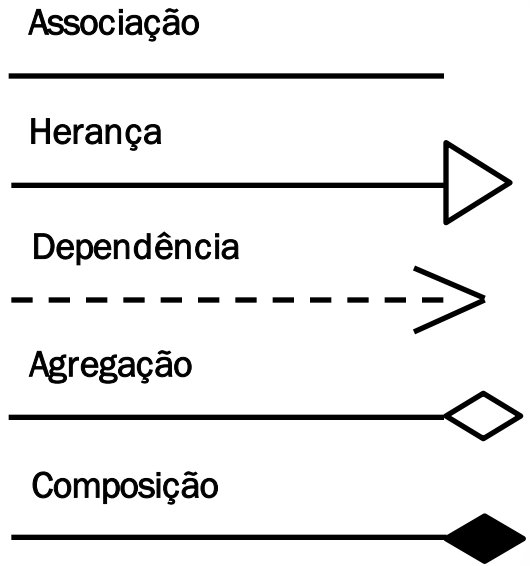
- Marcações de acesso podem ser usadas para especificar o tipo de acesso permitido aos atributos e métodos da classe.
  + \+ público
  + \# protegido
  + \- privado
  + Nenhuma marcação para default

Relacionamentos

- Permite compartilhar informações e colaborar com a execução dos processos do sistema.
  - Descreve um vínculo que ocorre normalmente entre os objetos de uma ou mais classes.
  - Os relacionamentos são representados por linhas ligando as classes envolvidas.

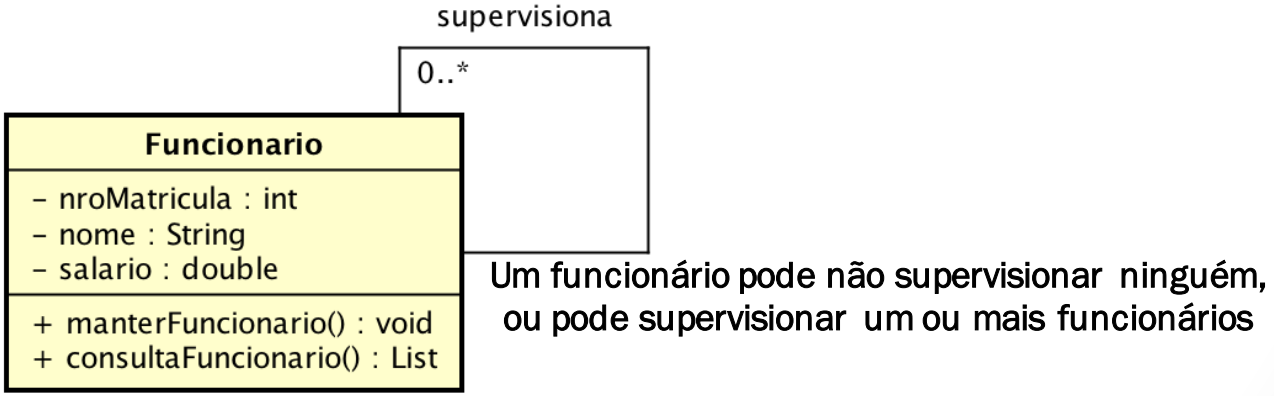
Tipos de Relacionamentos

- Os tipos de relacionamentos são:
  - Associação
    - Agregação
    - Composição
  - Especialização/Generalização
  - Dependência



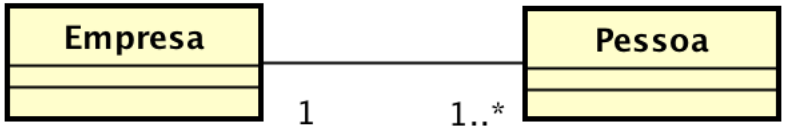
Associação Unária ou Reflexiva

- Ocorre quando existe um relacionamento de um objeto de uma classe com objetos da mesma classe.
  - No exemplo abaixo, um funcionário pode supervisionar outros funcionários.
  - O **supervisor** de um funcionário também é um **funcionário** da empresa e, portanto, também se constitui em uma instância da (



Multiplicidade

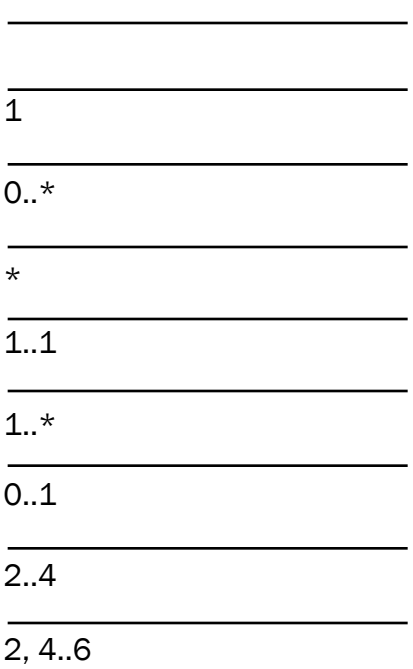
- Procura determinar o número mínimo e máximo de objetos envolvidos em cada extremidade da associação.
- Permite especificar o nível de dependência de um objeto para com os outros envolvidos na associação.



1 empresa possui 1 ou mais pessoas ligadas a ela
1 pessoa está ligada a apenas 1 empresa

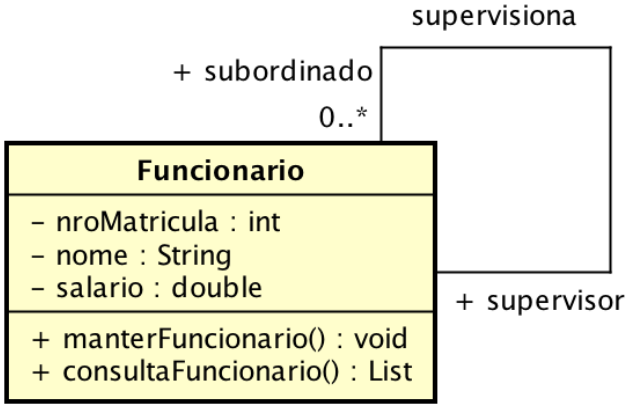
Tipos de Multiplicidade

- Não especificada
- Exatamente um
- Zero ou mais
- Muitos (mesmo que 0..*)
- Um e somente um
- Um ou mais
- Zero ou um
- Intervalo determinado
- Valores múltiplos



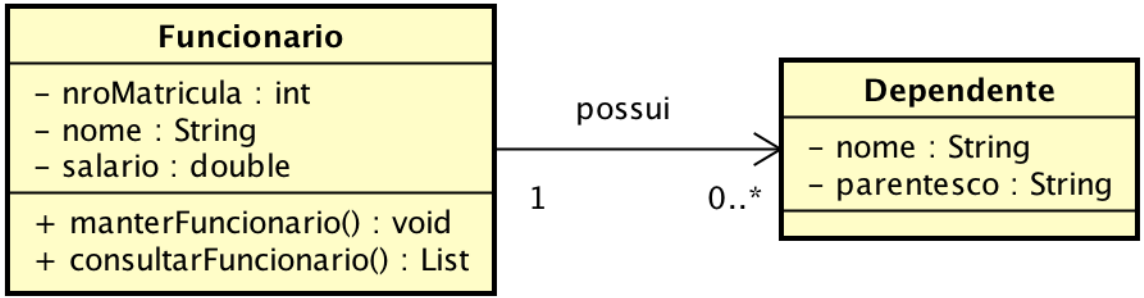
Papéis

- Informações extras na associação que podem ajudar a explicar a função de um objeto (o papel que este representa) dentro da associação.



Associação Binária

- Ocorrem quando são identificados relacionamentos entre objetos de duas classes distintas.

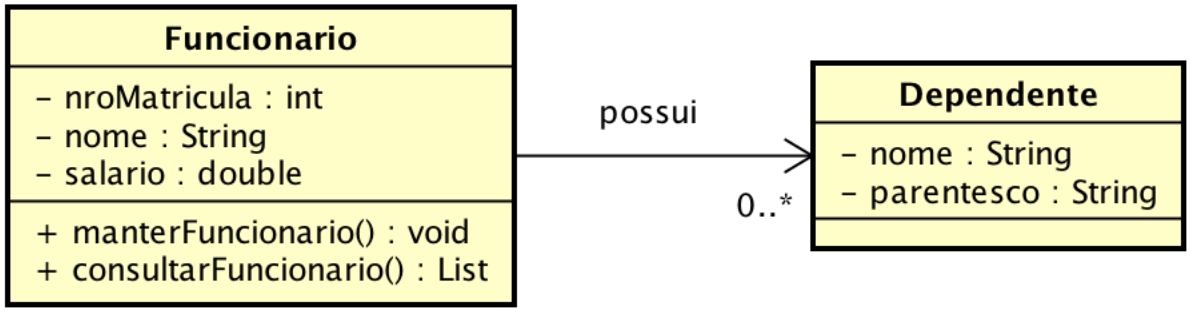



```
public class Funcionario {

        private int nroMatricula;
        private String nome;
        private double salario;
        private Dependente [] dependentes;

        //métodos
}
```

```
public class Dependente {

        private String nome;
        private String parentesco;
        //métodos
}
```

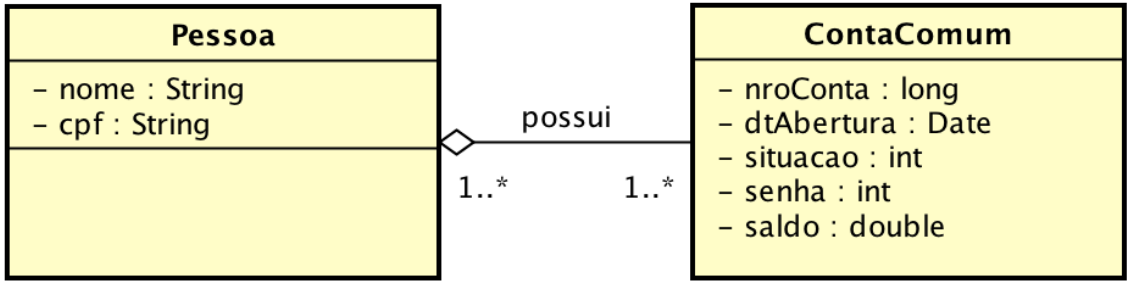
Navegabilidade

- É representada por uma seta em uma das extremidades e identifica o sentido em que as informações são transmitidas entre os objetos das classes envolvidas.
  - No exemplo abaixo, um objeto da classe Funcionario poderá chamar métodos da classe Dependente.



Agregação

- Demonstra que as informações de um objeto (objeto-**todo**) precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe (objetos-**parte**).
  - O símbolo de agregação contém um losango na extremidade da classe que contém os objetos-todo.



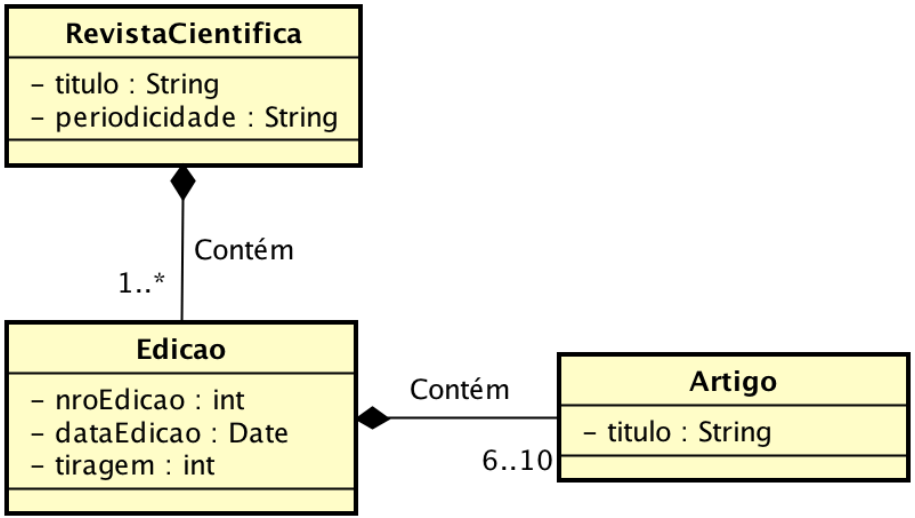
Objetos da classe **Pessoa** são **objetos-todo** que precisam ter suas informações complementadas pelos objetos da classe **ContaComum**, que nesta associação são **objetos-parte**.

Agregação

- A associação de agregação pode, em muitos casos ser substituída por uma associação binária simples.
  - A função principal de uma de uma associação do tipo agregação é identificar a obrigatoriedade de uma complementação das informações de um **objeto-todo** por seus **objetos-parte**, quando este for **consultado**.

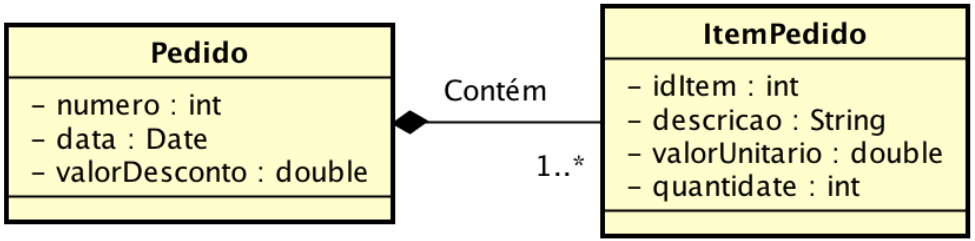
Composição

- É uma variação da agregação e considerada **mais forte**.
  - O objeto-parte **não** pode existir sem o objeto-todo.
  - Se o objeto-todo for destruído, o objeto-parte também será.



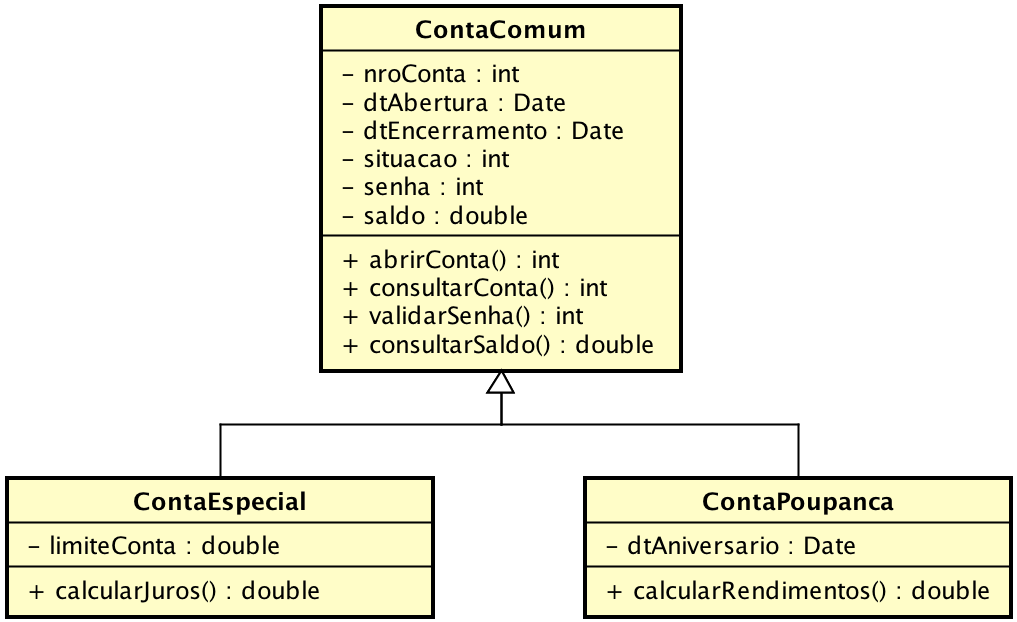
Composição

- É uma variação da agregação e considerada **mais forte**.
  - O objeto-parte **não** pode existir sem o objeto-todo.
  - Se o objeto-todo for destruído, o objeto-parte também será.



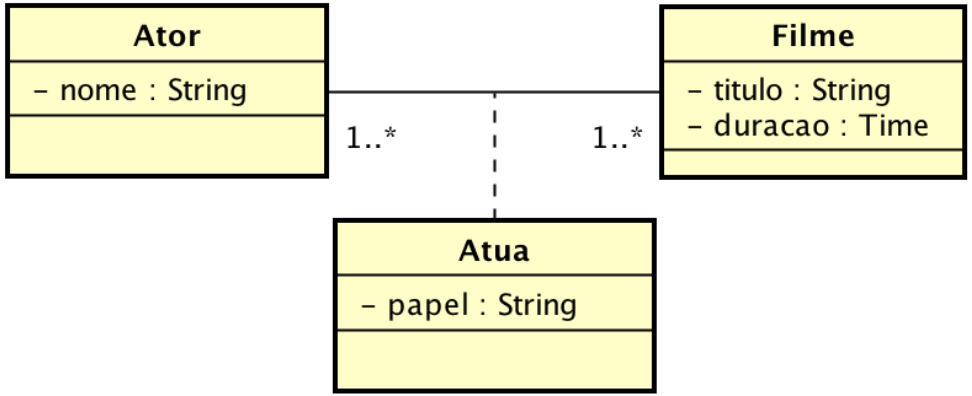
Generalização/Especialização

- Tem como objetivo representar a ocorrência de herança entre as classes identificando as superclasses e subclasses.



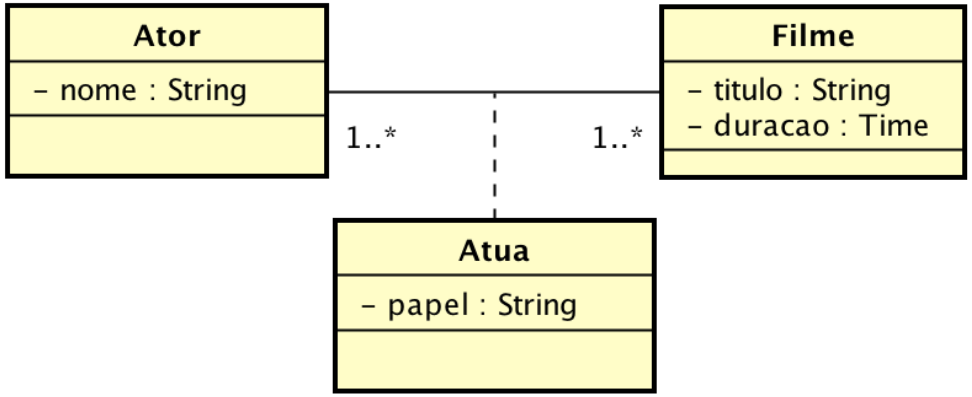
Classe Associativa

- Classes produzidas quando as associações possuem a multiplicidade muitos (*) em todas as suas extremidades.
  - São necessárias nos casos em que existem atributos relacionados à associação que **não podem** ser armazenados por nenhuma das classes envolvidas.



Classe Associativa

- Classes produzidas quando as associações possuem a multiplicidade muitos (*) em todas as suas extremidades.
  - São necessárias nos casos em que existem atributos relacionados à associação que **não podem** ser armazenados por nenhuma das classes envolvidas.



Um ator pode atuar em muitos filmes, e um filme pode ter muitos atores atuando nele.

Classe Associativa

Caso um ator interpretasse dois papéis em um mesmo filme?

Classe Associativa

Caso um ator interpretasse dois papéis em um mesmo filme?

O uso da classe associativa não seria o mais adequado.

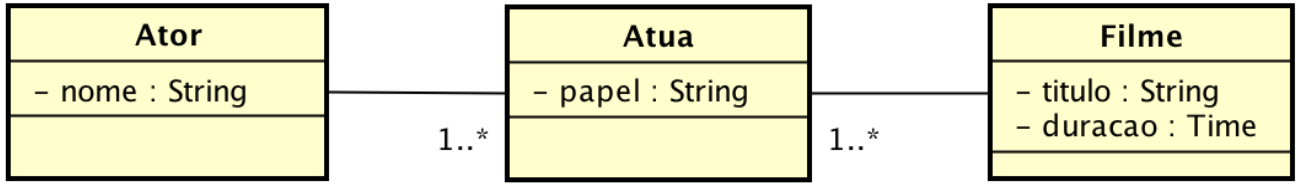
Poderíamos inserir uma classe normal atuando como uma ***classe intermediária****.*

Classe Associativa

Caso um ator interpretasse dois papéis em um mesmo filme?

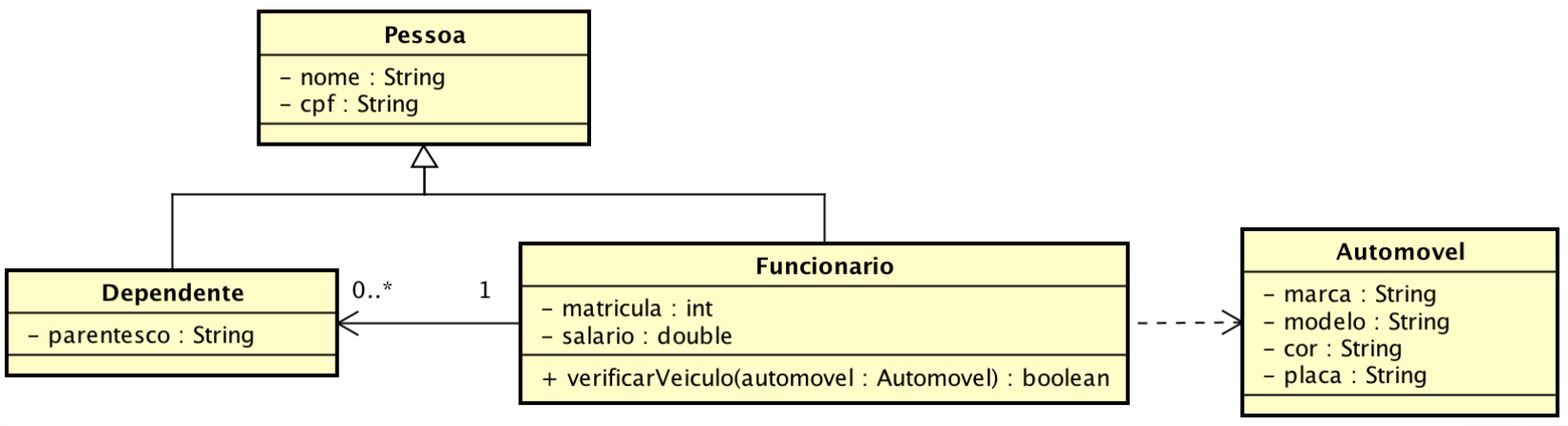
O uso da classe associativa não seria o mais adequado.

*Poderíamos inserir uma classe normal atuando como uma **classe intermediária**.*



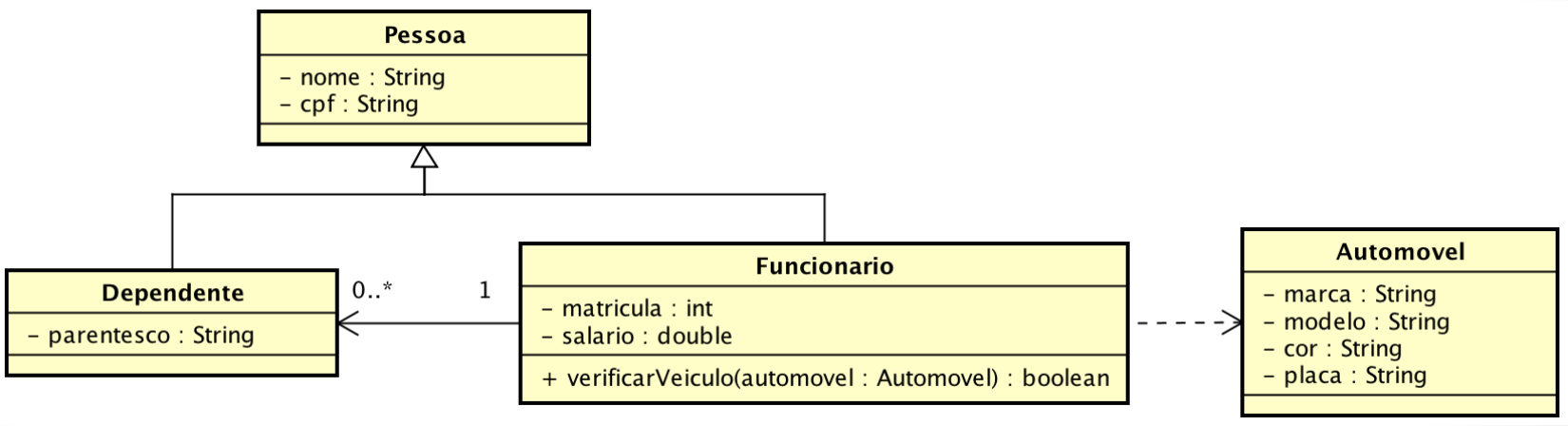
Dependência

- Identifica certo grau de dependência de uma classe em relação à outra.
  - Uma dependência difere de uma associação porque a conexão entre as classes é temporária.



Dependência

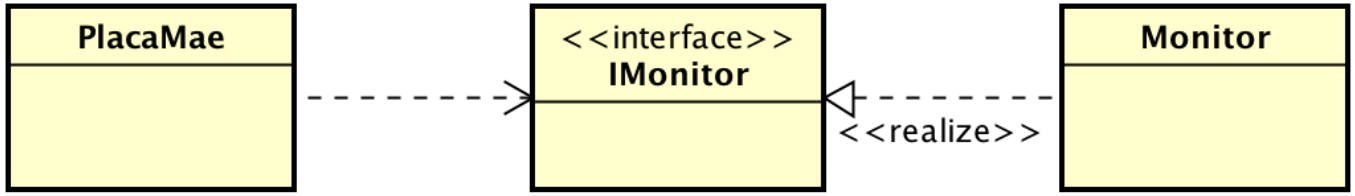
- Identifica certo grau de dependência de uma classe em relação à outra.
  - Uma dependência difere de uma associação porque a conexão entre as classes é temporária.



Funcionario não instancia um Automovel, apenas usa-o como parâmetro de um método

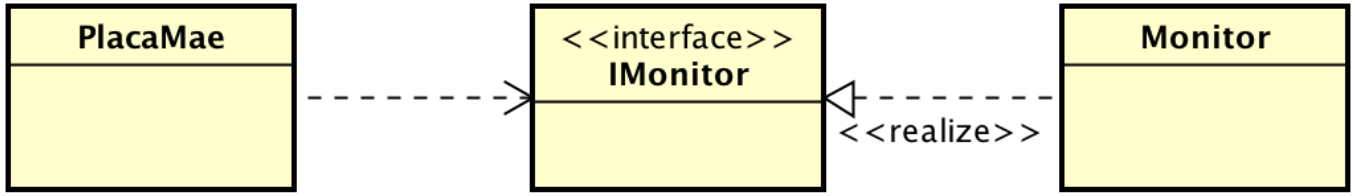
Realização

- Tipo de relacionamento especial que mistura características de generalização e dependência.
  - Usada para identificar classes responsáveis por executar funções para outras classes.



Realização

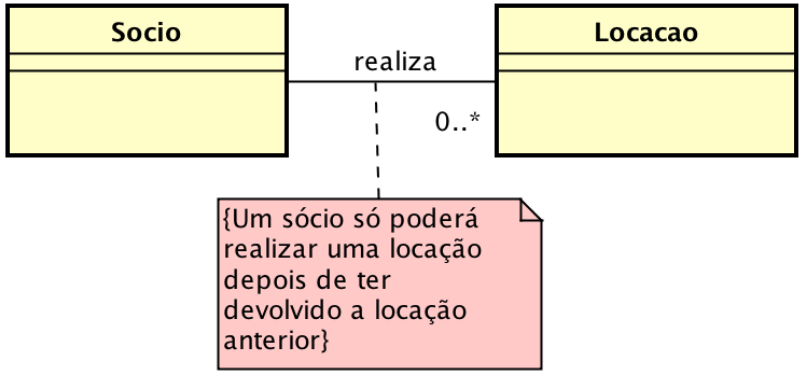
- Tipo de relacionamento especial que mistura características de generalização e dependência.
  - Usada para identificar classes responsáveis por executar funções para outras classes.



A classe Monitor implementa algum dos serviços oferecidos pela classe IMonitor.

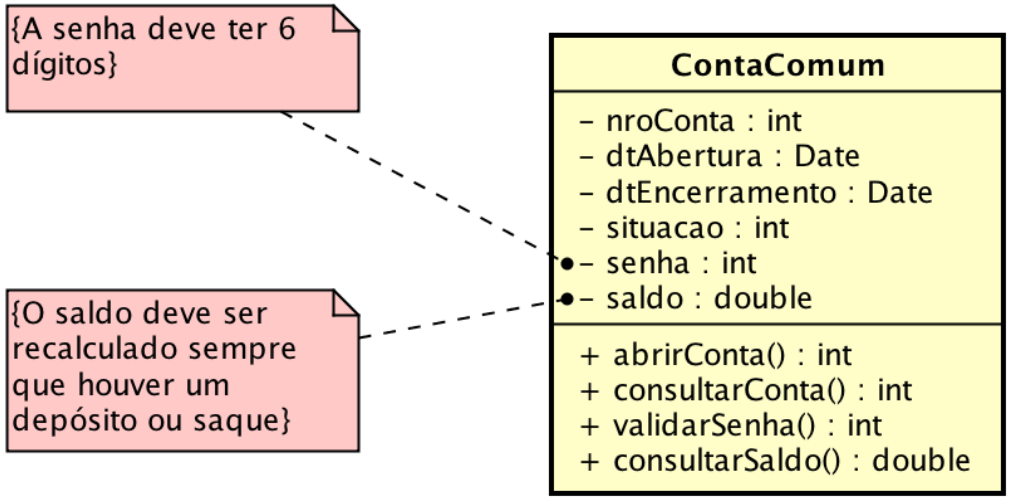
Restrição

- Informações extras que definem condições a serem validadas durante a implementação dos métodos de uma classe, das associações entre as classes ou mesmo de seus atributos.
  - Representadas por textos limitados por chaves.



Restrição

- Restrições podem ser aplicadas também para validar um atributo ou método de uma classe.

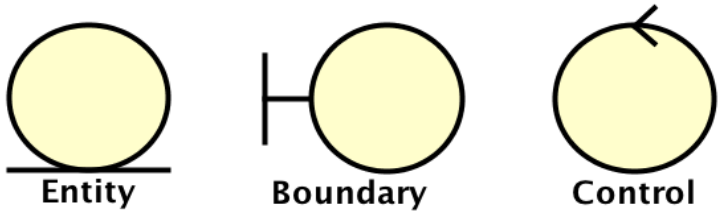


Estereótipos

- Possibilitam certo grau de extensibilidade aos componentes ou associações.
  - Permite a identificação de componentes ou associações que, embora semelhantes aos outros, tenham alguma característica que os diferenciem.
  - Existem 3 estereótipos predefinidos na UML bastante utilizados nos **diagramas de classes**:
    - <<entity>>
    - <<boundary>>
    - <<control>>

Classes Estereotipadas

- **Entity:** classe de entidade, geralmente implementa os objetos persistentes.
- **Boundary:** classe de fronteira, geralmente interfaces gráficas.
- **Control:** classe de controle, geralmente implementa as regras de negócio.

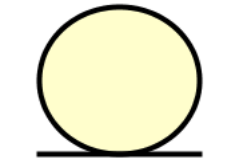


Estereótipo <<entity>>

- Tem por objetivo tornar explícito que uma classe é uma **entidade**.
  - Uma *entidade* é uma classe contém informações recebidas e armazenadas pelo sistema ou geradas por meio deste.
- Classes com o estereótipo <<*entity*>> também fornecem a informação de que normalmente terão muitos objetos, podendo ainda serem [illegible]s.

Estereótipo <<entity>>

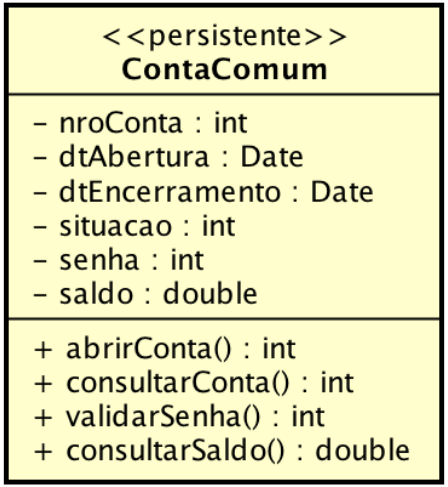
- Tem por objetivo tornar explícito que uma classe é uma **entidade**.
  - Uma *entidade* é uma classe contém informações recebidas e armazenadas pelo sistema ou geradas por meio deste.
- Classes com o estereótipo <<*entity*>> também fornecem a informação de que normalmente terão muitos objetos, podendo ainda serem [illegible]s.



Fica explícito que a classe ContaComum é uma entidade, ela armazena informações referentes ao problema que o sistema no qual ela está inserida procura solucionar.

Estereótipo Customizado

- No exemplo abaixo, definimos um novo estereótipo chamado <<*persistente*>> para deixar claro que a classe tem que preservar fisicamente suas instâncias.

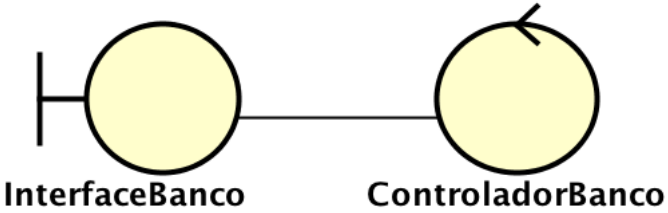


Estereótipo <<boundary>>

- Identifica uma classe que serve de comunicação entre os atores externos e o sistema propriamente dito.
  - Muitas vezes uma classe <<*boundary*>> é associada à própria interface do sistema.
  - Uma classe do tipo <<*boundary*>> muitas vezes necessita interagir com outra classe do tipo <<*control*>>
  - Sua utilização é importante quando é preciso definir a existência de uma interface para o sistema.

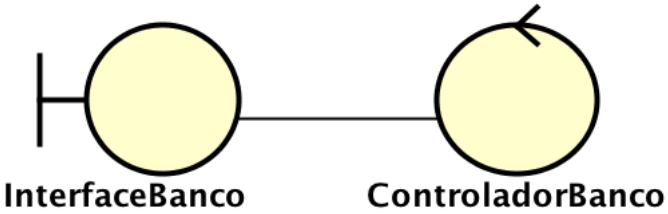
Estereótipo <<control>>

- Identifica classes que servem de **intermédio** entre as classes <<*boundary*>> e as demais classes do sistema.
  - Os objetos <<*control*>> são responsáveis por interpretar os eventos ocorridos sobre os objetos <<*boundary*>> e retransmiti-los aos objetos das classes de *entidade* que fazem parte do sistema.



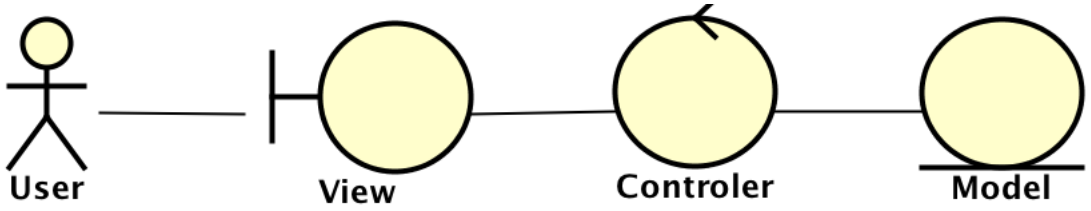
Estereótipo <<control>>

- Identifica classes que servem de **intermédio** entre as classes <<*boundary*>> e as demais classes do sistema.
  - Os objetos <<*control*>> são responsáveis por interpretar os eventos ocorridos sobre os objetos <<*boundary*>> e retransmiti-los aos objetos das classes de *entidade* que fazem parte do sistema.

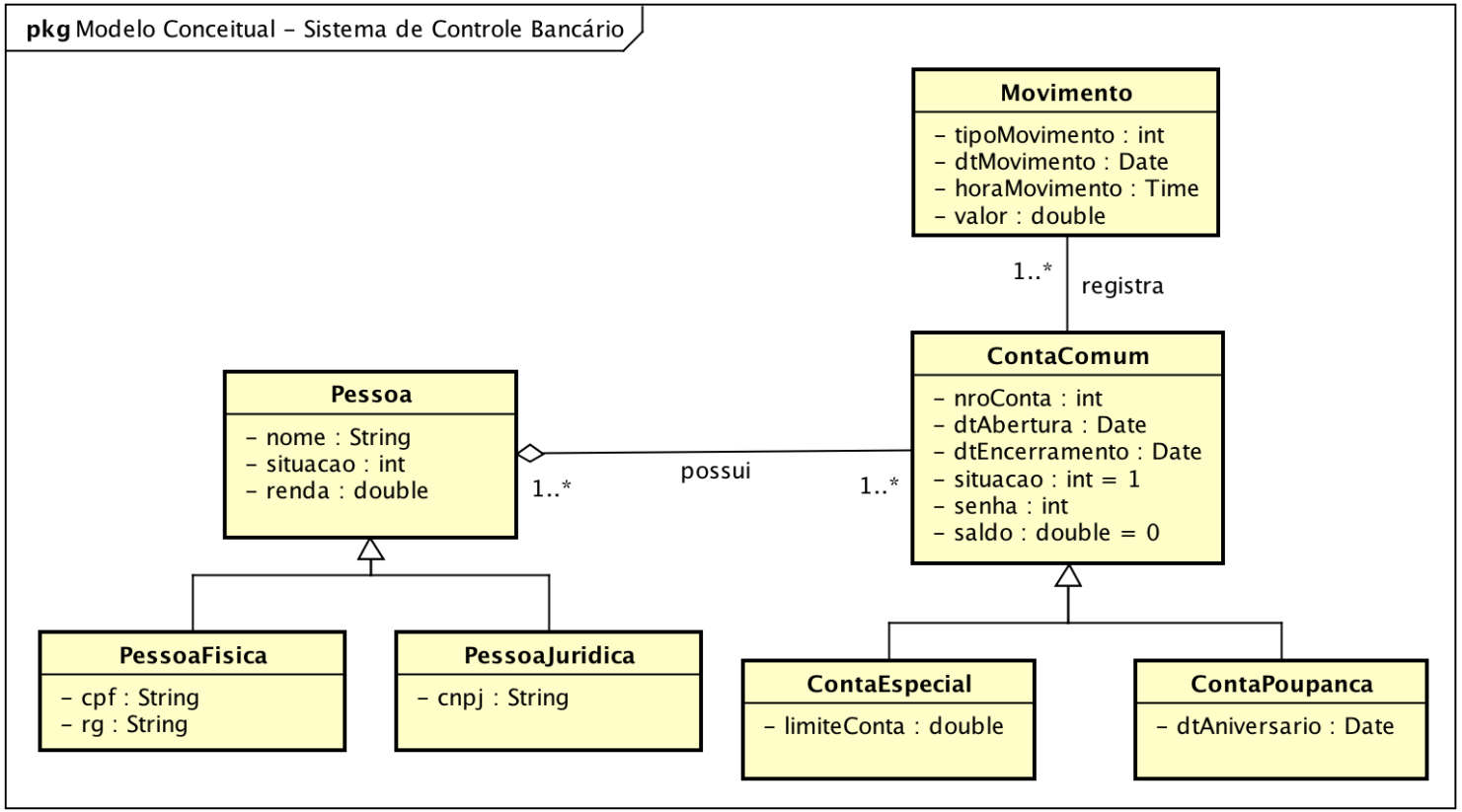


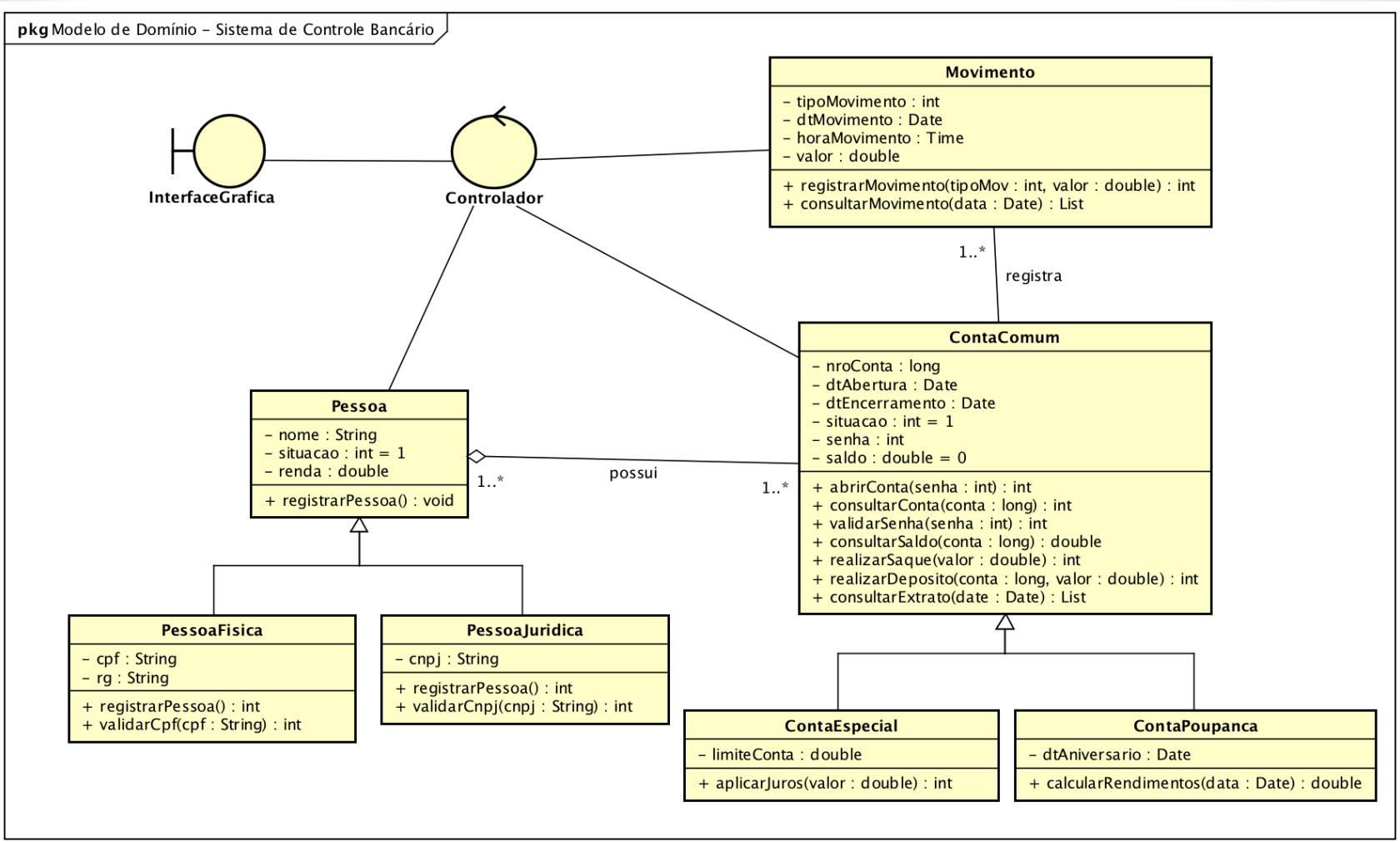
A classe InterfaceBanco representa a interface do sistema e seus objetos representam os componentes gráficos, os eventos que ocorrem sobre tais objetos são repassados para um objeto da classe ControladorBanco que interpretará esses eventos.

Boundary, Control e Entity



Exemplo de Diagrama de Classes




pkg Modelo de Domínio – Sistema de Controle Bancário
InterfaceGrafica
Controlador
Movimento
– tipoMovimento : int
– dtMovimento : Date
– horaMovimento : Time
– valor : double
+ registrarMovimento(tipoMov : int, valor : double) : int
+ consultarMovimento(data : Date) : List
1..*
registra
ContaComum
– nroConta : long
– dtAbertura : Date
– dtEncerramento : Date
– situacao : int = 1
– senha : int
– saldo : double = 0
+ abrirConta(senha : int) : int
+ consultarConta(conta : long) : int
+ validarSenha(senha : int) : int
+ consultarSaldo(conta : long) : double
+ realizarSaque(valor : double) : int
+ realizarDeposito(conta : long, valor : double) : int
+ consultarExtrato(date : Date) : List
Pessoa
– nome : String
– situacao : int = 1
– renda : double
+ registrarPessoa() : void
1..*
possui
1..*
PessoaFisica
– cpf : String
– rg : String
+ registrarPessoa() : int
+ validarCpf(cpf : String) : int
PessoaJuridica
– cnpj : String
+ registrarPessoa() : int
+ validarCnpj(cnpj : String) : int
ContaEspecial
– limiteConta : double
+ aplicarJuros(valor : double) : int
ContaPoupanca
– dtAniversario : Date
+ calcularRendimentos(data : Date) : double

Lista de Exercícios II

UML e Padrões de Projeto - Lista de Exercícios II.pdf

60min

Bibliografia

- GUEDES, Gilleanes. UML Uma Abordagem Prática. Editora Novatec. São Paulo, 2014.
- FURLAN, José. Modelagem de Objetos através da UML. Editora Makron Books.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML Guia do Usuário. Editora Campus.