Diagrama de Classes

Prof. Ma. Marina Girolimetto

marina.girolimetto@uffs.edu.br

Diagrama de Classes

- **Seu principal enfoque está em permitir a visualização das classes que comporão o sistema com seus respectivos atributos e métodos, bem como em demonstrar como as classes do diagrama se relacionam, complementam e transmitem informações entre si.**
- Esse diagrama apresenta uma visão estática de como as classes estão organizadas, preocupando-se em como definir a estrutura lógica delas.

Diagrama de Classes

- **O diagrama de classes serve ainda como apoio para a construção da maioria dos outros diagramas da linguagem UML.**
- Recomenda-se que se utilize o diagrama de classes ainda durante a fase de análise, produzindo-se um modelo conceitual onde o engenheiro preocupa-se apenas em representar as classes, seus atributos e as associações entre as classes, não modelando características como os métodos.

Diagrama de Classes

- Somente na fase de projeto produz-se o modelo de domínio, que já enfoca a solução do problema.
- **Os métodos necessários às classes são descobertos a partir da modelagem de diagramas de interação, como o diagrama de sequência.**

- Um projeto pode ser dividido em subsistemas e cada um deles pode possuir seus diagramas de classes particulares. Essas aplicações do diagrama de classes dependerão do enfoque e do objetivo com que será aplicado.

Atributos e Métodos

- Classes costumam ter **atributos que armazenam os dados dos objetos da classe.**
- Classes também costumam possuir **métodos**, também chamados operações, **que são as funções que uma instância da classe pode executar**.
- Os valores dos atributos podem variar de uma instância para outra, ao passo que os métodos são idênticos para todas as instâncias de uma classe específica.
- Embora os métodos sejam declarados no diagrama de classes, o diagrama de classes não se preocupa em definir as etapas que tais métodos deverão percorrer quando forem chamados, sendo essa uma função atribuída a outros diagramas, como o diagrama de atividade.

Atributos e Métodos

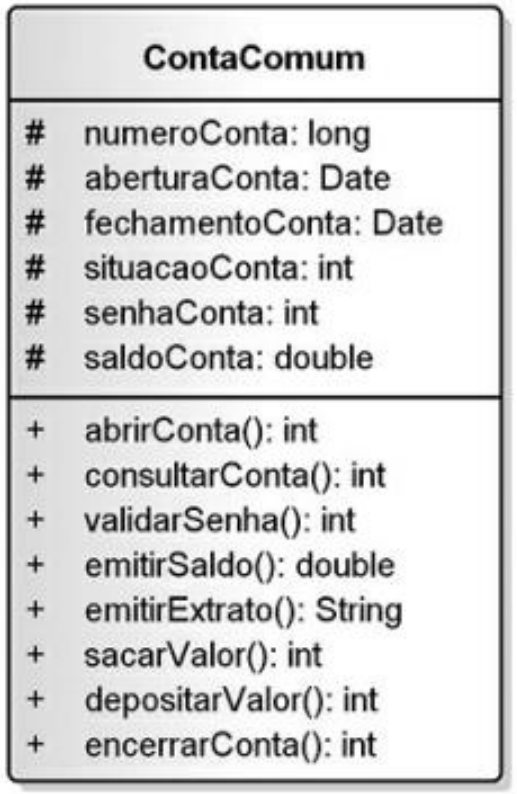


Figura 4.1 – Classe.

- Uma classe, na linguagem UML, é representada como um retângulo com até três divisões descritas a seguir:
  - **A primeira contém a descrição ou nome da classe.**
  - **A segunda armazena os atributos e seus tipos de dados.**
  - **Finalmente, a terceira divisão lista os métodos da classe.**
- Os símbolos de sustenido (#) e mais (+) na frente dos atributos e métodos representam a visibilidade destes, o que determina quais objetos de quais classes podem utilizar o atributo ou o método em questão.

Atributos e Métodos

- **Não é realmente obrigatório que uma classe apresente as três divisões.**
- Métodos podem receber valores como parâmetros e retornar valores que podem ser o resultado produzido pela execução do método ou simplesmente um valor representado se o método foi realizado com sucesso ou não, por exemplo.
- O detalhamento dos parâmetros nos métodos é opcional.

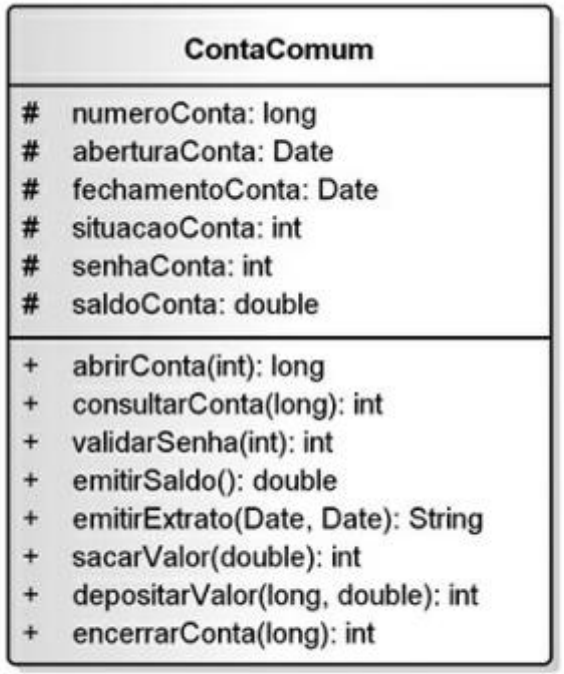


Figura 4.2 – Detalhamento das Assinaturas das Operações.

- **Código correspondente a essa classe implementado em Java:**

```
public class ContaComum {
  protected long numeroConta;
  protected Date aberturaConta;
  protected Date fechamentoConta;
  protected int situacaoConta;
  protected int senhaConta;
  protected double saldoConta;

  public ContaComum(){
  }

  public long abrirConta(int senha){
    return 0;
  }

  public int consultarConta(long numeroConta){
    return 0;
  }

  public int validarSenha(int senha){
    return 0;
  }

  public double emitirSaldo(){
    return 0;
  }

  public String emitirExtrato(Date dataInicial, Date dataFinal){
    return "";
  }

  public int sacarValor(double valor){
    return 0;
  }

  public int depositarValor(long numeroConta, double valor){
    return 0;
  }

  public int encerrarConta(){
    return 0;
  }
}
```

Atributos e Métodos

- Os atributos de uma classe podem ainda ter características extras: **valores iniciais, multiplicidade e se o atributo é derivado**, ou seja, se seus valores são produzidos por meio de algum tipo de cálculo.

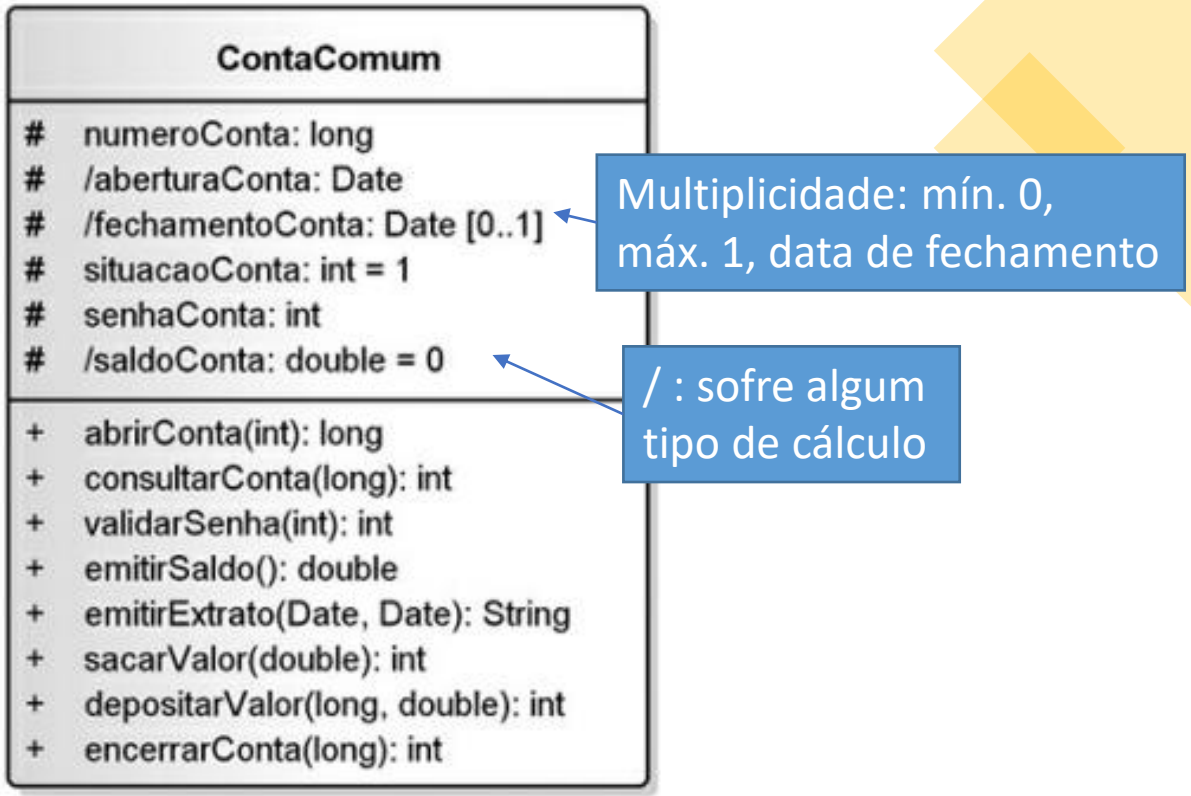


Figura 4.3 – Detalhamento dos Atributos.

Relacionamentos ou Associações

- **As classes costumam ter relacionamentos entre si, chamados <u>associações</u>**, que permitem que elas compartilhem informações entre si e colaborem para a execução dos processos executados pelo sistema.
- **Uma associação descreve um vínculo que ocorre normalmente entre os objetos de uma ou mais classes.**
- As associações são representadas por linhas ligando as classes envolvidas. Tais linhas podem ter nomes ou títulos para auxiliar a compreensão do tipo de vínculo estabelecido entre os objetos das classes envolvidas nas associações.

Associação Unária ou Reflexiva

- **Este tipo de associação ocorre quando existe um relacionamento de um objeto de uma classe com objetos da mesma classe.**

multiplicidade 0.. demonstra que um determinado funcionário pode chefiar nenhum (0) ou muitos (*) funcionários*

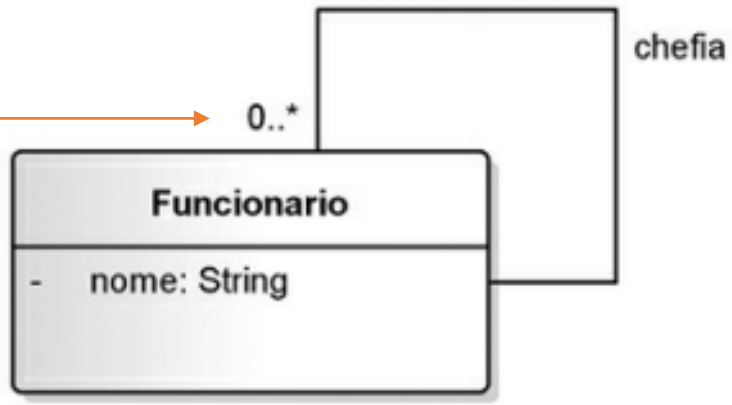


Figura 4.4 – Associação Unária.

<- não é obrigatório informar

Quando não existe multiplicidade explícita, assume-se que é 1..1

Tabela 4.1 – Exemplos de multiplicidade

Multiplicidade	Significado
0..1	No mínimo, zero (nenhum) e, no máximo, um. Indica que os objetos das classes associadas não precisam obrigatoriamente estar relacionados, mas se houver relacionamento, indicará que apenas uma instância da classe relaciona-se com as instâncias da outra classe (ou da outra extremidade da associação, se esta for unária).
1..1	Um e somente um. Indica que apenas um objeto da classe relaciona-se com os objetos da outra classe.
0..*	No mínimo, nenhum e, no máximo, muitos. Indica que pode ou não haver instâncias da classe participando do relacionamento.
*	Muitos. Indica que muitos objetos da classe estão envolvidos na associação.
1..*	No mínimo, um e, no máximo, muitos. Indica que há pelo menos um objeto envolvido no relacionamento, podendo haver muitos objetos envolvidos.
3..5	No mínimo, três e, no máximo, cinco. Estabelece que existem pelo menos três instâncias envolvidas no relacionamento, mas podem ser quatro ou cinco as instâncias envolvidas, mas não mais do que isso.

Associação Unária ou Reflexiva

- Outra informação é a **definição de papéis**, visto que se trata de uma informação extra na associação que pode ajudar a explicar a função de um objeto (o papel que este representa) dentro da associação.

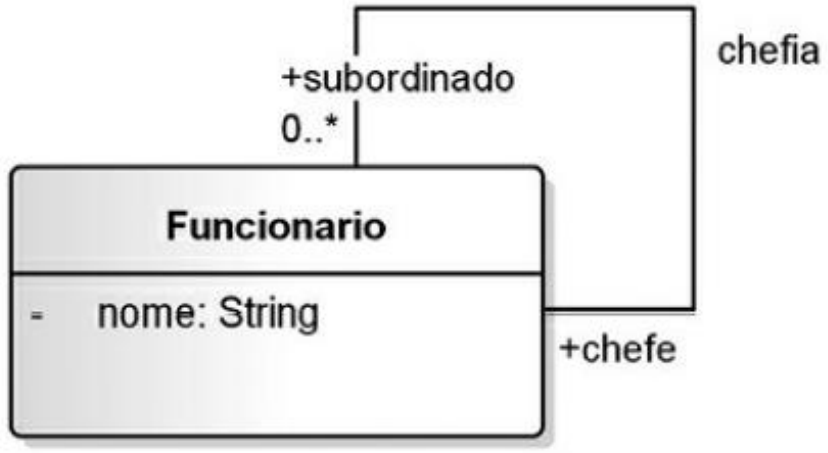


Figura 4.5 – Associação Contendo Papéis.

Associação Binária

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

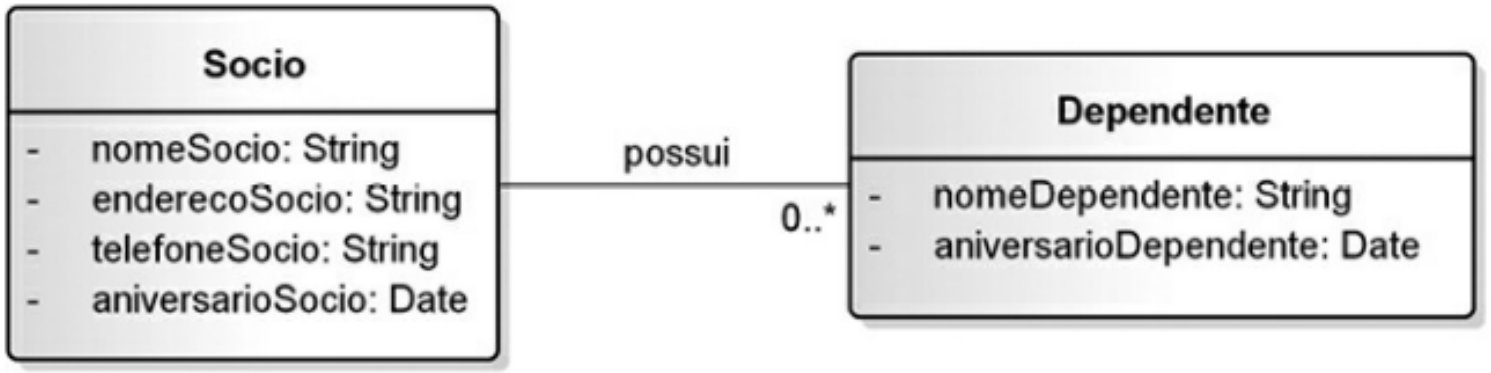


Figura 4.6 – Associação Binária.

Associação Binária

- Poderíamos acrescentar outras informações a essa associação, como definir a **navegabilidade** dela.
- **A navegabilidade é representada mais comumente por uma seta em um dos fins da associação**, embora seja possível representá-la nos dois sentidos.

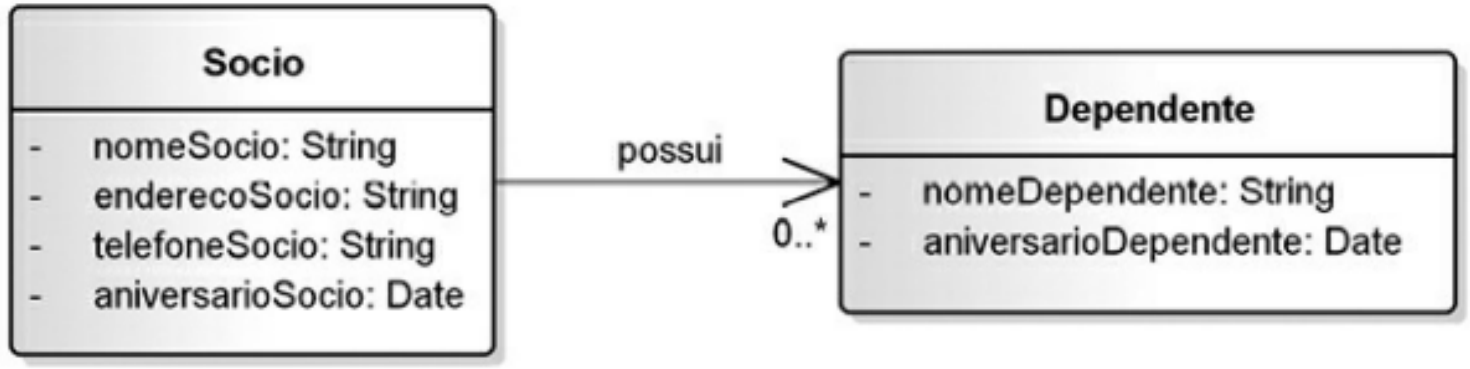


Figura 4.7 – Associação binária com Navegabilidade.

Associação Binária

- **A navegabilidade também determina o sentido em que os métodos poderão ser disparados**. Nesse exemplo, um objeto da classe Socio poderá disparar métodos em objetos da classe Dependente, mas a recíproca não é verdadeira: um objeto da classe Dependente não poderá disparar métodos em um objeto da classe Socio.
- **A navegabilidade não é obrigatória**, mesmo porque, se não houver setas, significará que as informações podem trafegar entre os objetos de todas as classes da associação.

Associação Ternária ou N-ária

- **Conectam objetos de mais de duas classes.** São representadas por um losango para onde convergem todas as ligações da associação.

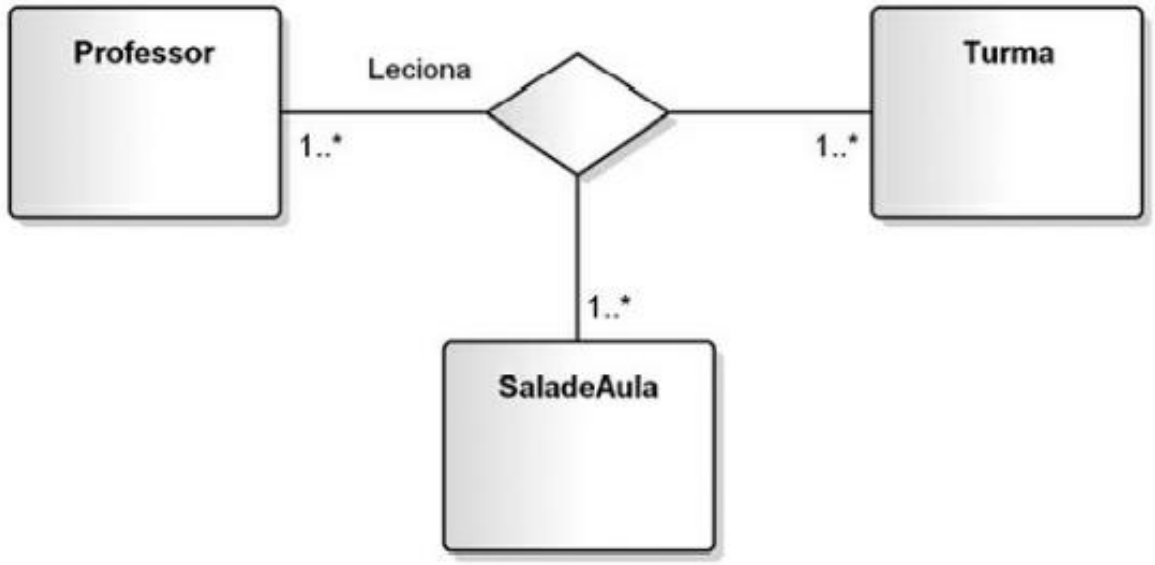
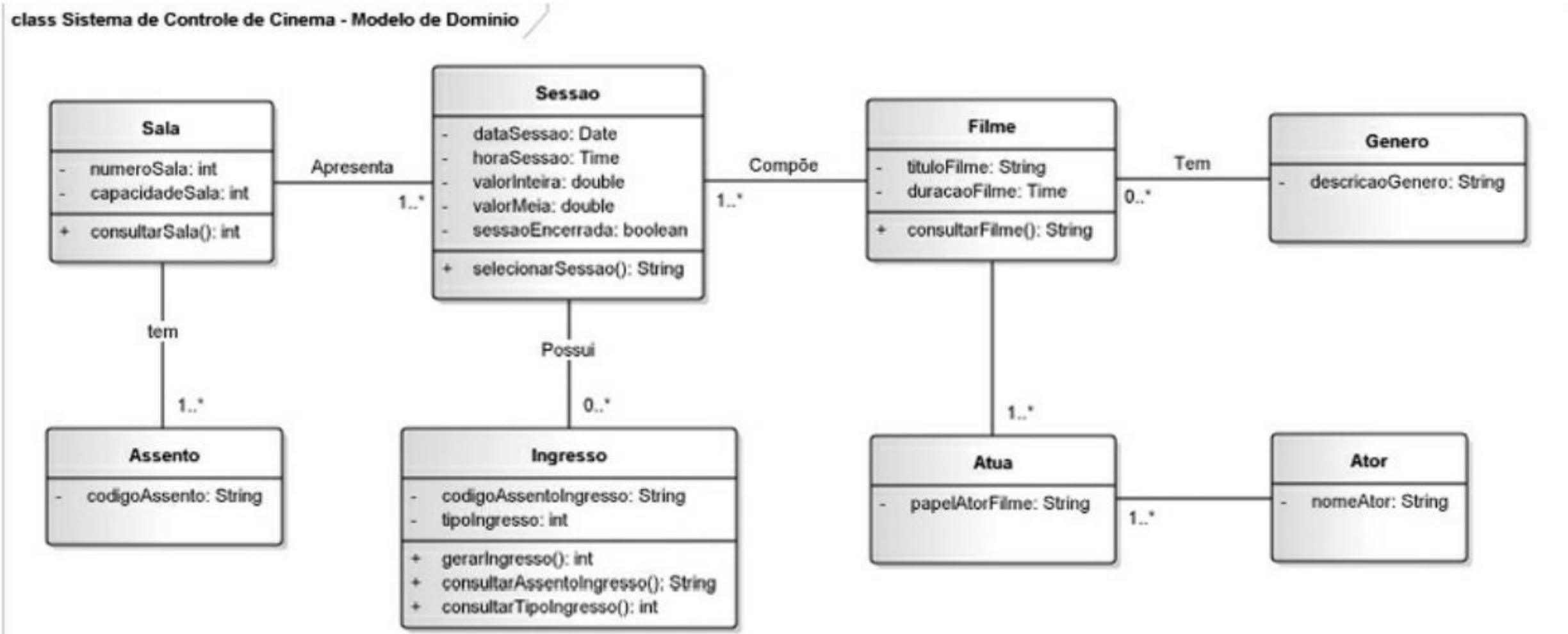


Figura 4.9 – Associação Ternária.

Exercício - Sistema de Controle de Cinema

- **Desenvolva o diagrama de classes:**
- Um cinema pode ter muitas salas, sendo necessário, portanto, registrar informações a respeito de cada sala, como sua capacidade, ou seja, o número de assentos disponíveis.
- O cinema apresenta muitos filmes. Um filme tem informações como título e duração. Assim, sempre que um filme for apresentado, deve-se registrá-lo também.
- Um filme tem um único gênero, mas um gênero pode se referir a muitos filmes.
- Um filme pode ter muitos atores atuando nele e um ator pode atuar em muitos filmes.
- Em cada filme, um ator interpretará um ou mais papéis diferentes. Por uma questão de propaganda, é útil anunciar os principais atores do filme e que papéis eles interpretam.
- Um mesmo filme pode ser apresentado em diferentes salas e horários. Cada apresentação em uma determinada sala e horário é chamada Sessão. Um filme apresentado em uma sessão tem um conjunto máximo de ingressos, determinado pela capacidade da sala.
- Os clientes do cinema podem comprar ou não ingressos para assistir a uma sessão. O funcionário deve intermediar a compra do ingresso. Um ingresso deve conter informações como o tipo de ingresso (meia-entrada ou ingresso inteiro). Além disso, um cliente só pode comprar ingressos para sessões ainda não encerradas.

Resolução


class Sistema de Controle de Cinema - Modelo de Dominio
Sala
- numeroSala: int
- capacidadeSala: int
+ consultarSala(): int
Apresenta
1..*
Sessao
- dataSessao: Date
- horaSessao: Time
- valorInteira: double
- valorMeia: double
- sessaoEncerrada: boolean
+ selecionarSessao(): String
Compõe
1..*
Filme
- tituloFilme: String
- duracaoFilme: Time
+ consultarFilme(): String
Tem
0..*
Genero
- descricaoGenero: String
tem
1..*
Assento
- codigoAssento: String
Possui
0..*
Ingresso
- codigoAssentoIngresso: String
- tipoIngresso: int
+ gerarIngresso(): int
+ consultarAssentoIngresso(): String
+ consultarTipoIngresso(): int
1..*
Atua
- papelAtorFilme: String
1..*
Ator
- nomeAtor: String

Resolução

- Gênero – Essa classe armazena os gêneros de filmes apresentados pelo cinema. Seu único atributo é a descrição do gênero do tipo String.
- Filme – Essa classe contém o título e a duração de cada filme apresentado no cinema. Seu único método serve para consultar os filmes cadastrados. Observe que um filme está associado a um único gênero, mas um gênero pode estar associado a muitos filmes.
- Ator – Essa classe representa os atores que interpretam papéis nos filmes exibidos no cinema. Seu único atributo é o nome do ator.
- Atua – Esta é uma classe intermediária entre as classes Ator e Filme e representa os papéis que um ator interpreta em cada filme de que participa. Seu único atributo consiste no papel que um ator interpreta. Observe que um ator pode atuar em, no mínimo, um e, no máximo, muitos filmes e que um filme pode ter muitos atores atuando nele, mas um filme deve ter pelo menos um ator.
- Sala – Essa classe armazena as informações sobre as salas pertencentes ao cinema. Seus atributos são número da sala e capacidade da sala, ambos do tipo inteiro. Seu único método permite consultar uma determinada sala e retornar sua capacidade

Resolução

- Assento – Essa classe serve apenas para armazenar os códigos de assento de uma determinada sala.
- Sessão – Essa classe representa as sessões de filmes apresentadas pelo cinema. Seus atributos são data da sessão, hora da sessão, valor do ingresso inteiro, valor da meia-entrada e um atributo para determinar se a sessão já foi encerrada ou não. Observe que uma sessão é apresentada em uma única sala, mas em uma sala podem ter sido apresentadas muitas sessões. Note também que em uma sessão é exibido somente um filme, mas um filme pode ser exibido em muitas sessões. O único método definido nessa classe permite selecionar as sessões ainda em aberto, retornando uma String com os dados da sessão em questão.
- Ingresso – Finalmente, essa classe representa os ingressos vendidos em cada sessão de cinema. Essa classe armazena o código do assento a que se refere o ingresso e o tipo de ingresso, se é um inteiro ou meia-entrada. A classe possui ainda os seguintes métodos:

Resolução

Método	Descrição
`gerarIngresso`	Gera um novo ingresso, retornando verdadeiro (1), se foi possível gerar o ingresso, ou falso (0), caso contrário. Observe que uma sessão pode gerar muitos ingressos, contudo pode não gerar ingresso algum, conforme demonstra a multiplicidade.
`consultarAssentoIngresso`	Retorna uma String contendo código do assento de um determinado ingresso.
`consultarTipoIngresso`	Retorna o tipo de ingresso de um ingresso específico.

Agregação

- **Tipo especial de associação em que se tenta demonstrar que as informações de um objeto (objeto-todo) são complementadas pelas informações contidas em um ou mais objetos no outro fim da associação (chamados objetos-parte).**
- O símbolo de agregação difere do de associação por conter um losango no fim da associação que contém os objetos-todo.

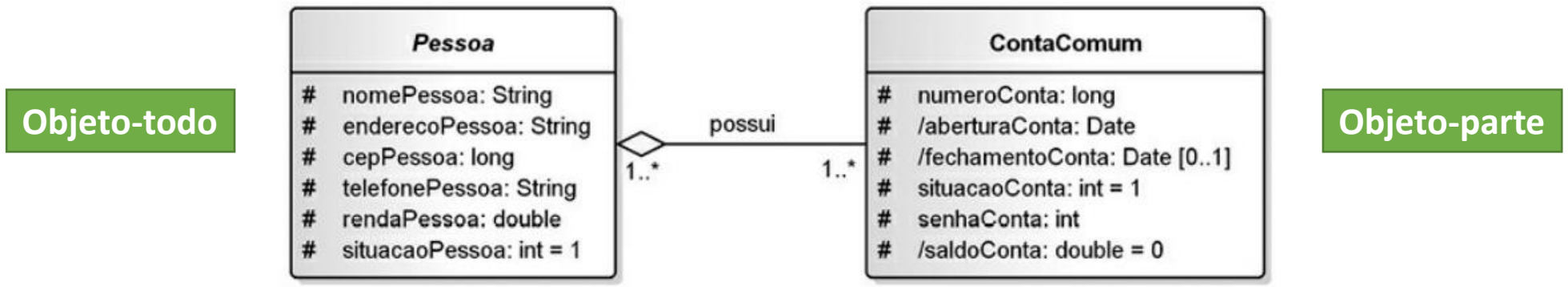


Figura 4.10 – Agregação.

Composição

- **Variação da agregação**, onde é apresentado um vínculo mais forte entre os objetos-todo e os objetos-parte, procurando demonstrar que os objetos-parte têm de estar associados a um único objeto-todo.
- Da mesma forma que na agregação, o losango deve ficar ao lado do objeto-todo.

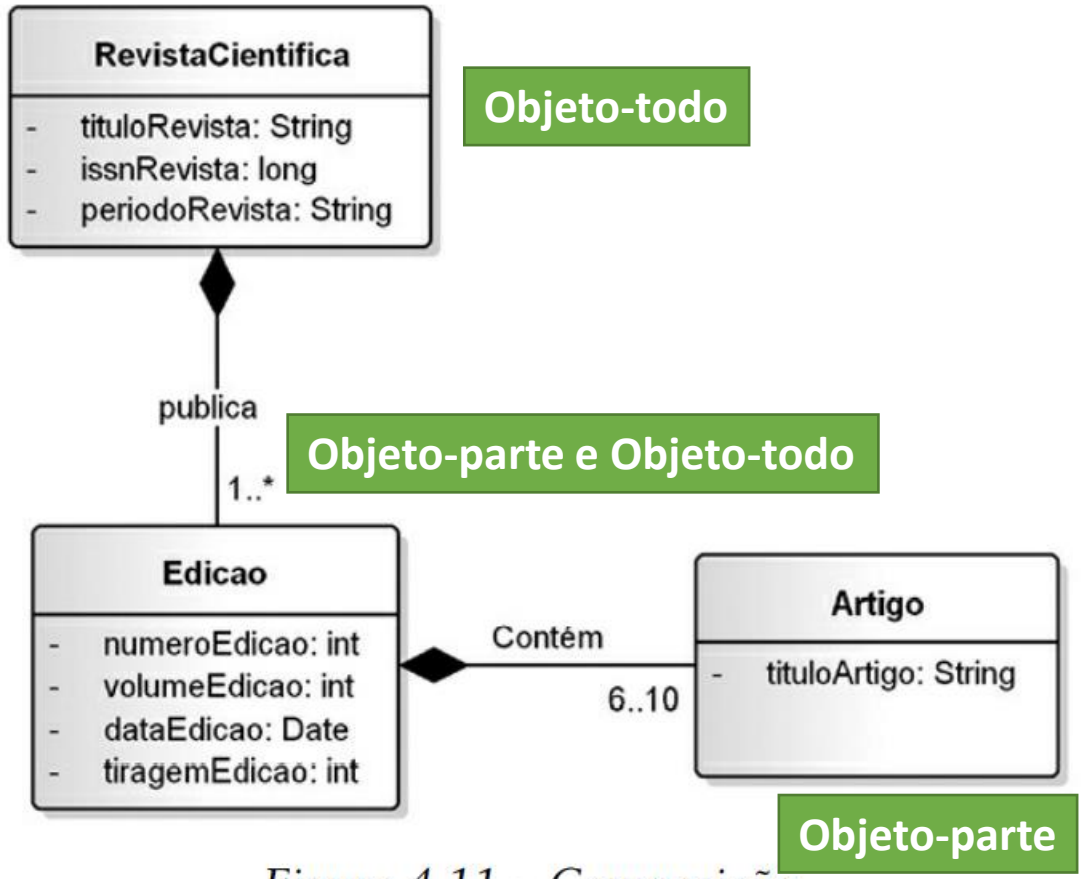


Figura 4.11 – Composição.

Generalização/Especialização

- Similar à associação de mesmo nome utilizada no diagrama de casos de uso.
- **O objetivo dessa associação é representar a hierarquia entre as classes e, possivelmente, métodos polimórficos nas classes especializadas**.
- O símbolo de generalização/especialização é o mesmo do diagrama de casos de uso.

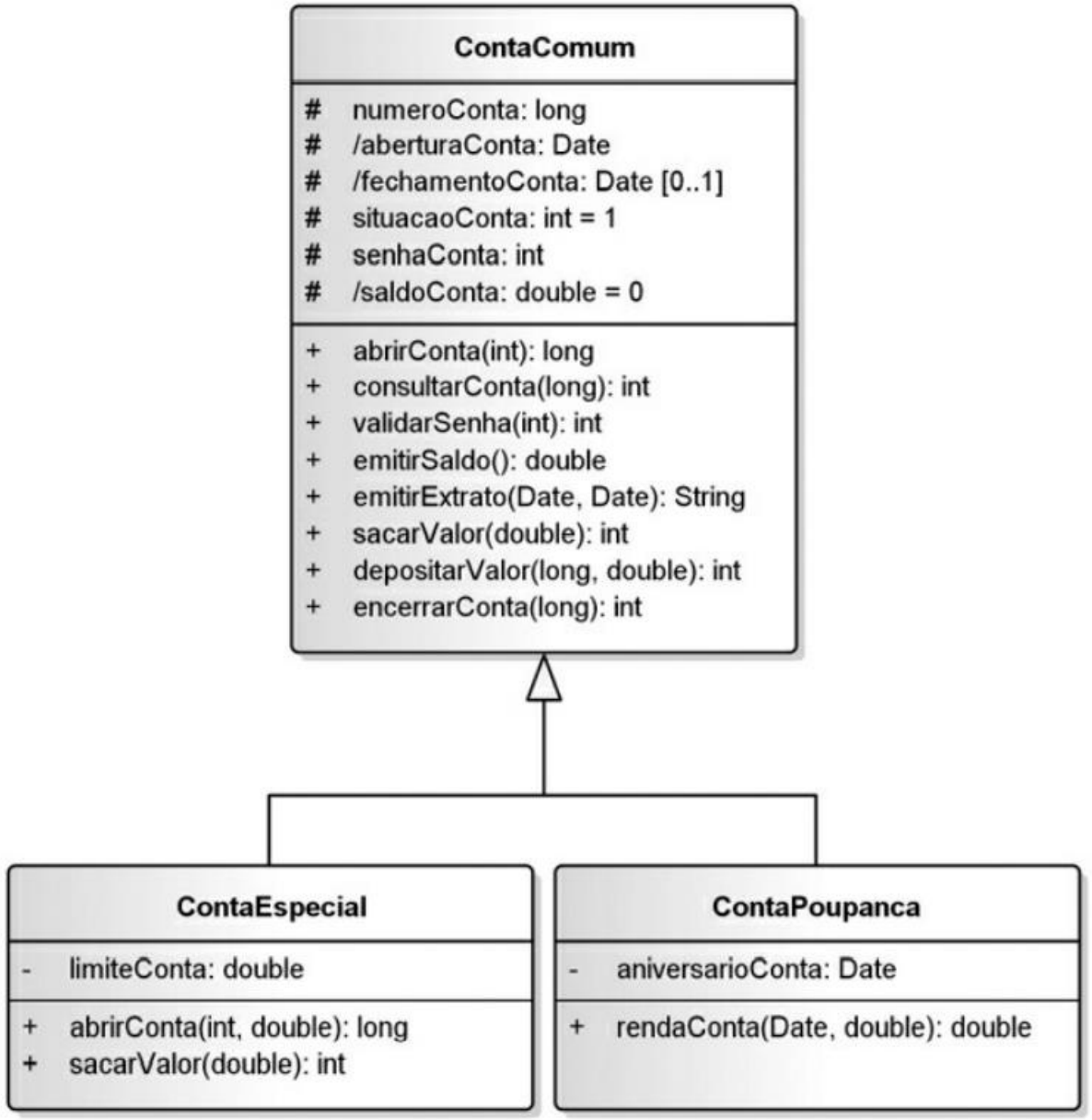


Figura 4.12 – Generalização/Especialização no Diagrama de Classes.

Classe Associativa

- **São necessárias nos casos em que existem atributos relacionados à associação que não podem ser armazenados por nenhuma das classes envolvidas.**
- Utilizadas principalmente em associações que apresentem multiplicidade muitos (*).

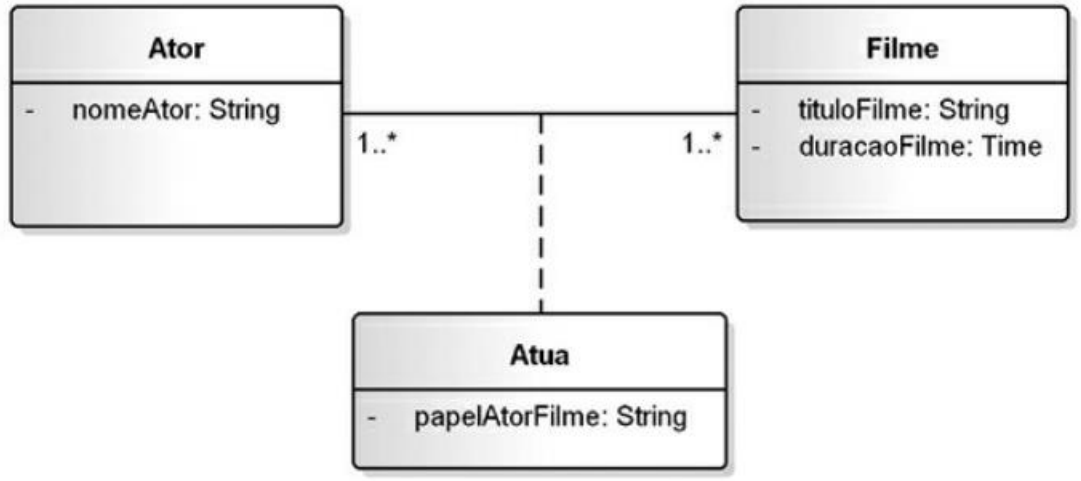


Figura 4.13 – Classe Associativa.

Classe Associativa

- Classes associativas são válidas somente quando existe um único objeto relacionado a duas instâncias associadas. No caso deste exemplo, **um ator que atue em um filme terá um único papel**. Caso um ator interpretasse dois papéis em um mesmo filme, o uso da classe associativa não seria o mais adequado, sendo necessário inserir uma classe normal atuando como classe intermediária da associação.

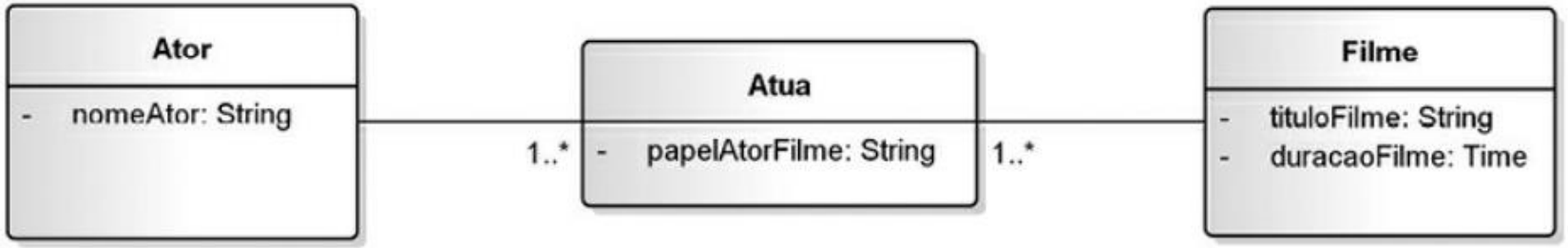


Figura 4.14 – Classe Intermediária.

Associação Qualificada

- Quando existe um atributo único em uma classe, é possível criar uma associação qualificada, que é **uma forma de identificar individualmente um objeto dentro de uma coleção (uma coleção pode ser definida como um conjunto de instâncias associadas a outro objeto)**. O qualificador de uma associação é representado por um pequeno retângulo ligado ao final da associação.

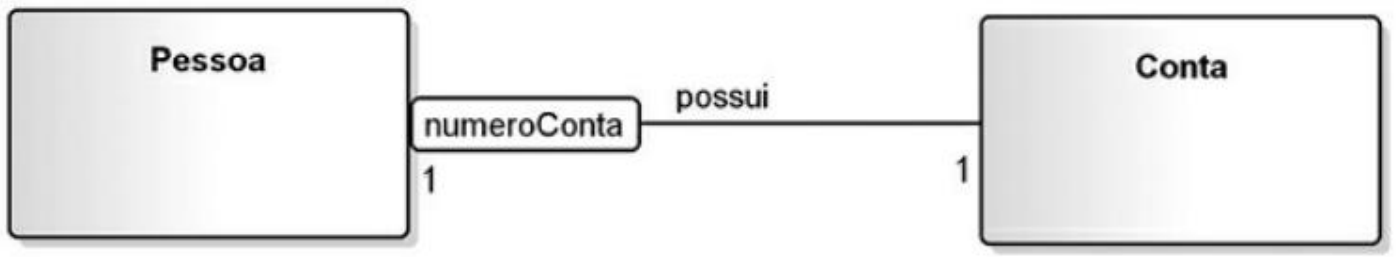


Figura 4.15 – Exemplo de Associação Qualificada.

Reduz a multiplicidade: 1 para n; p/ 1 para 1

Dependência

- **Identifica certo grau de dependência de um elemento (normalmente uma classe) em relação à outro**.
- O relacionamento de dependência possui muitas especializações utilizadas como estereótipos, por exemplo, *usage* (uso) é um tipo de dependência no qual um elemento necessita que um ou mais elementos o complementem de alguma forma, como em uma operação, por exemplo.

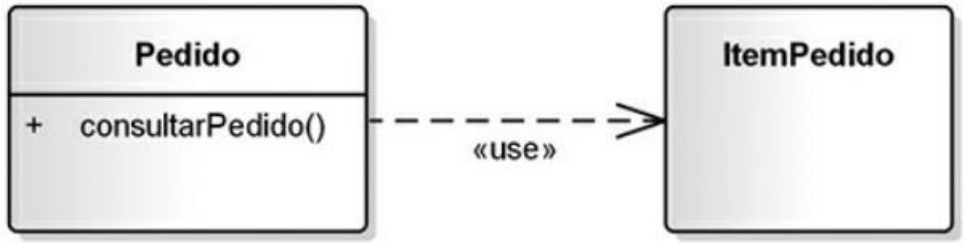


Figura 4.16 – Dependência com Estereótipo Usage.

Realização

- **É um tipo de dependência especializada que mistura características dos relacionamentos de generalização e dependência**, sendo usada para identificar classes responsáveis por executar funções para outras classes. Esse tipo de relacionamento herda o comportamento de uma classe, mas não sua estrutura.

Figura 4.17 – Relacionamento de Dependência e Realização.

Comparada a *implements* da linguagem Java, que determina que uma classe implementará os métodos definidos em outra classe.

Realização

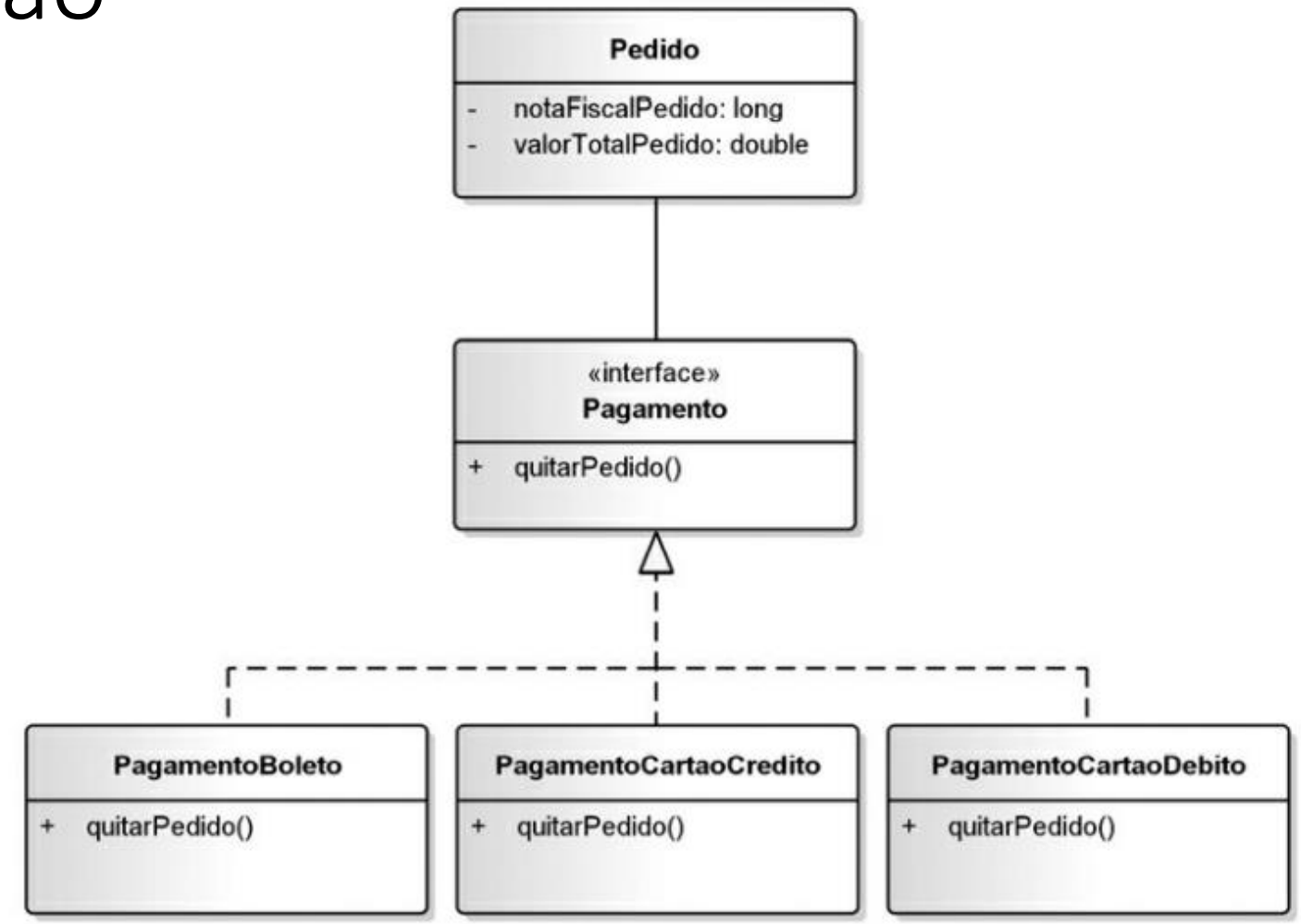
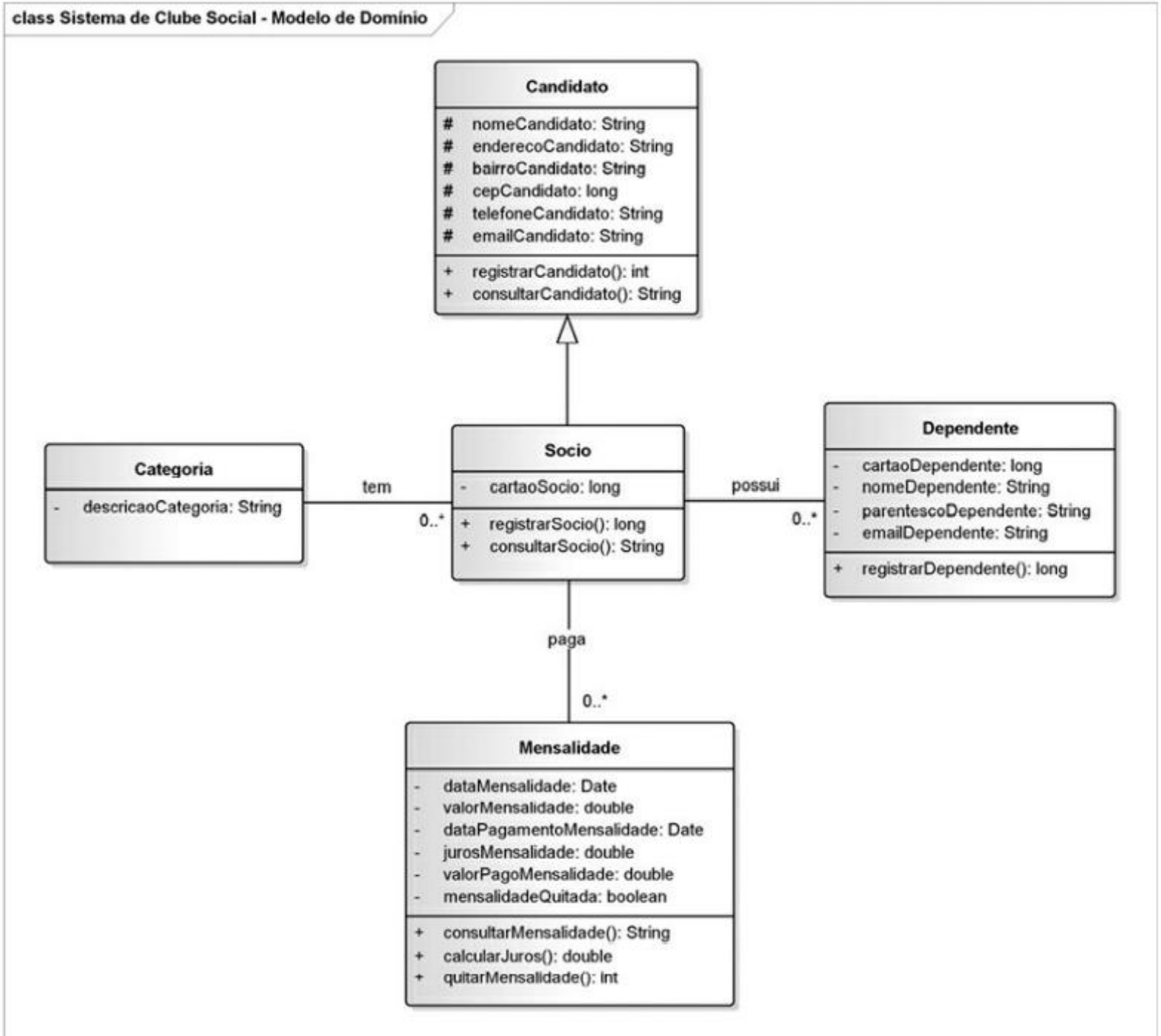


Figura 4.18 – Relacionamento de Realização.

Exercício - Sistema de Controle de Clube Social

- **Desenvolva o diagrama de classes:**
- O clube tem muitos sócios e precisa manter informações referentes a eles, como o número de seu cartão de sócio, nome, endereço, telefone e e-mail.
- Um sócio pode ter nenhum ou muitos dependentes, mas um dependente está associado a somente um sócio. O clube precisa manter informações sobre os dependentes de cada sócio, como o número de seu cartão, nome, parentesco e e-mail.
- Um sócio deve pertencer a uma única categoria. No entanto, pode haver muitos sócios pertencentes a uma determinada categoria.
- Um sócio deve pagar mensalidades para poder frequentar o clube. Assim, enquanto permanecer sócio do clube, um sócio poderá pagar muitas mensalidades, mas uma mensalidade pertence a somente um sócio. Eventualmente, um sócio pode não estar adimplente. Nesse caso, serão cobrados juros sobre o valor da mensalidade relativos ao atraso do pagamento. É também possível que um sócio nunca tenha pago suas mensalidades. As informações pertinentes a cada mensalidade são a data de pagamento, o valor, a data em que foi efetivamente paga, os possíveis juros aplicados, o valor efetivamente pago e se está quitada ou não.

Resolução


class Sistema de Clube Social - Modelo de Domínio
Candidato
nomeCandidato: String
enderecoCandidato: String
bairroCandidato: String
cepCandidato: long
telefoneCandidato: String
emailCandidato: String
+ registrarCandidato(): int
+ consultarCandidato(): String
Categoria
- descricaoCategoria: String
tem
0..*
Socio
- cartaoSocio: long
+ registrarSocio(): long
+ consultarSocio(): String
possui
0..*
Dependente
- cartaoDependente: long
- nomeDependente: String
- parentescoDependente: String
- emailDependente: String
+ registrarDependente(): long
paga
0..*
Mensalidade
- dataMensalidade: Date
- valorMensalidade: double
- dataPagamentoMensalidade: Date
- jurosMensalidade: double
- valorPagoMensalidade: double
- mensalidadeQuitada: boolean
+ consultarMensalidade(): String
+ calcularJuros(): double
+ quitarMensalidade(): int

Resolução

- Candidato – Essa classe armazena os dados das pessoas que se candidatam a sócios do clube. Os atributos dessa classe são autoexplicativos. Observe que a visibilidade dos atributos é protegida, posto que essa classe é especializada pela classe Socio. A classe tem dois métodos, um para registrar um candidato e outro para consultar um candidato.
- Socio – Essa classe armazena as informações referentes aos sócios do clube. Os atributos dessa classe são todos herdados da classe Candidato, exceto o atributo referente ao número do cartão do sócio que só existirá se um candidato vier a ser aprovado. A classe tem dois métodos, um para registrar um sócio e outro para consultar um sócio específico. O método registrarSocio retorna um long que representa o número do cartão do sócio e o método consultarSocio retorna uma String contendo os dados de um determinado sócio. Observe que um sócio pertence a uma categoria, mas uma categoria pode estar associada a muitos sócios.

Resolução

- Dependente – Essa classe armazena as informações referentes aos possíveis dependentes de um sócio. Os atributos da classe são autoexplicativos. O único método contido pela classe permite gerar uma nova instância dela. Pode-se perceber que um dependente está relacionado a um único sócio, mas um sócio pode não ter nenhum dependente ou pode ter vários.
- Categoria – Essa classe representa as possíveis categorias de sócios estabelecidas pelo clube. Seu único atributo é a descrição da categoria. Pode haver muitos sócios associados a uma determinada categoria, por outro lado pode não haver nenhum sócio associado a uma categoria específica.

Resolução

- Mensalidade – A classe em questão representa as mensalidades que devem ser pagas por cada sócio. Seus atributos são data da mensalidade e data em que a mensalidade foi efetivamente paga, do tipo Date, valor da mensalidade, juros da mensalidade, valor efetivamente pago do tipo double e um atributo denominado mensalidadeQuitada do tipo boolean, que determina se a mensalidade foi quitada ou não. Já os métodos da classe são:

Método	Descrição
`consultarMensalidade`	É disparado para consultar cada mensalidade ainda não paga de um determinado sócio. Retorna uma `String` com os dados da mensalidade.
`calcularJuros`	Calcula os juros de uma mensalidade, no caso de esta estar atrasada. Retorna um `double` contendo o valor atual, após a aplicação de juros, da mensalidade.
`quitarMensalidade`	Permite a quitação de uma mensalidade, retornando verdadeiro, se a operação foi concluída com sucesso, ou falso, se ocorreu algum problema quando se tentou quitar a mensalidade.