benefícios de uma camada de modelo e impede a aplicação de técnicas orientadas a objetos que permitem organizar uma lógica complexa.

Ao utilizar o padrão MVC, adotamos a diretriz que as classes controladoras na camada de controle devem ser magras e as classes de entidade na camada de modelo devem ser gordas, no sentido que devem conter o comportamento relativo à lógica do negócio. Contudo, nada impede que o leitor adote outra abordagem, porém, se quiser utilizar os modelos descritos neste livro, terá que transferir métodos da camada de modelo para a camada de controle, o que pode ser um pouco difícil em algumas situações e necessitar de certas adaptações.

4.7 Exemplo de Diagrama de Classes (Modelo Conceitual) – Sistema de Controle Bancário

Nesta seção, modelaremos o modelo conceitual do diagrama de classes para o sistema de controle bancário iniciado no capítulo 3. Conforme foi explicado, o modelo conceitual é produzido durante a fase de análise de requisitos e refere-se ao domínio do problema, enquanto o modelo de domínio é produzido durante a fase de projeto e refere-se ao domínio da solução. O modelo conceitual apresenta somente as informações que o sistema necessitará, enquanto o modelo de domínio toma o modelo conceitual e detalha questões como métodos, navegabilidade e até mesmo pode inserir novas classes, se isso for considerado necessário.

Ao longo deste estudo de caso, apresentaremos os dois modelos, enquanto nos exemplos seguintes apresentaremos somente o modelo de domínio. É importante destacar que o modelo de domínio somente deve ser desenvolvido na fase de projeto e deve ser modelado com o detalhamento dos casos de uso por meio de diagramas de interação, como o de sequência, onde se percebe quais métodos serão necessários em cada processo, enriquecendo-se, assim, o modelo de domínio.

Cumpre ainda destacar que apesar de haver recursos já descritos como agregação, composição, generalização/especialização ou classes associativas, não é obrigatória sua utilização em todo diagrama de classes. Esses recursos deverão ser utilizados eventualmente, quando for necessário. A figura 4.49 ilustra o modelo conceitual para o sistema de controle bancário.

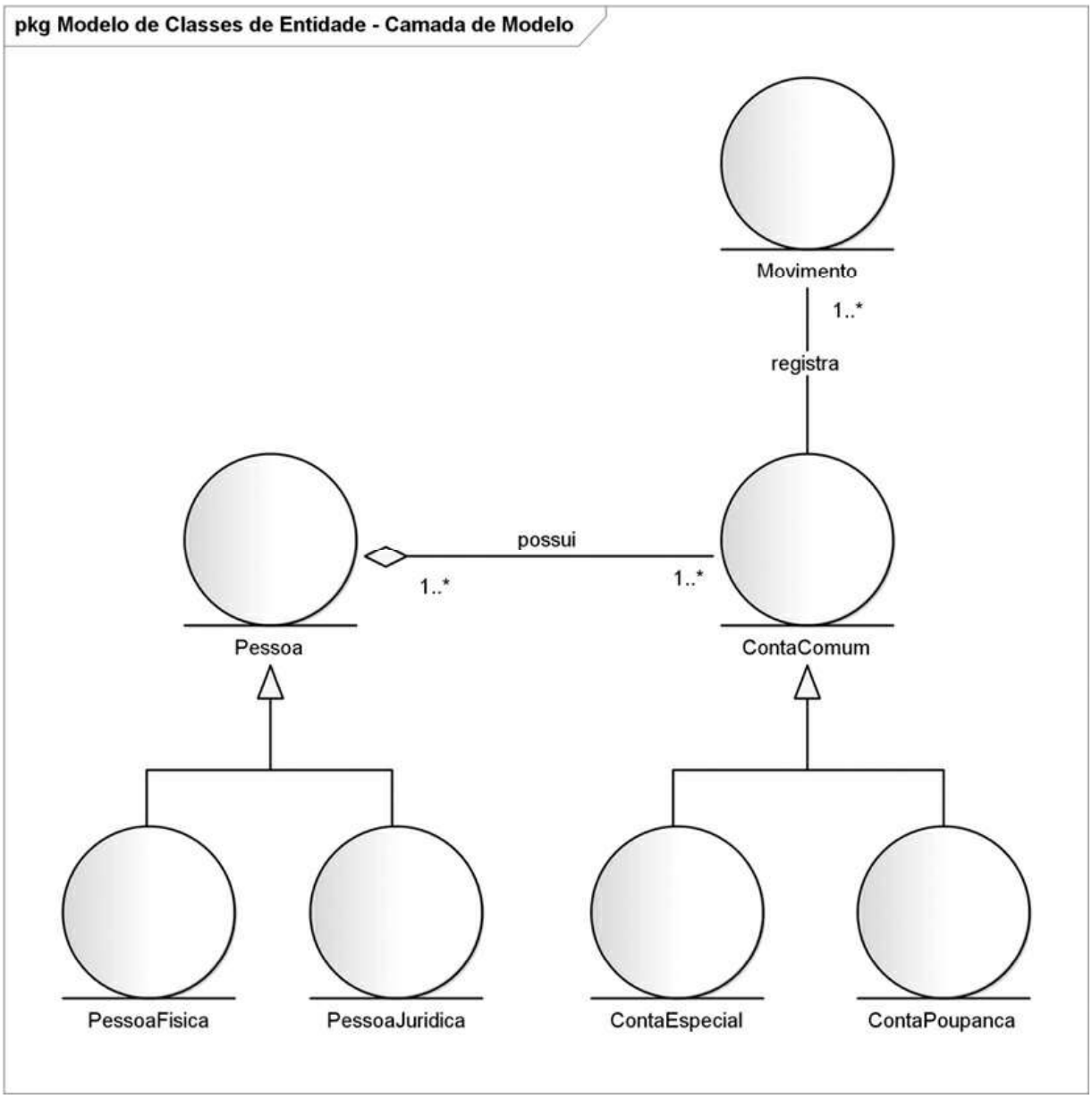

Figura 4.49 – Diagrama de Classes para o Sistema de Controle Bancário – Modelo Conceitual – Uso do Estereótipo Entity.

Nesta figura, as classes foram representadas com o estereótipo **<<entity>>**, que, como foi dito, é um estereótipo gráfico que modifica o desenho-padrão do componente. A vantagem de utilizar esse estereótipo é a de declarar explicitamente que se trata de uma classe de entidade e, em modelos grandes, diminuir o espaço do diagrama. Contudo, preferimos utilizar a imagem-padrão de uma classe porque assim podem ser vistos os atributos e métodos que ela contém. Nos próximos modelos, utilizaremos

a imagem-padrão para poder ilustrar os atributos e métodos de cada classe. A figura 4.50 apresenta o mesmo modelo, porém sem utilizar estereótipos de entidade.

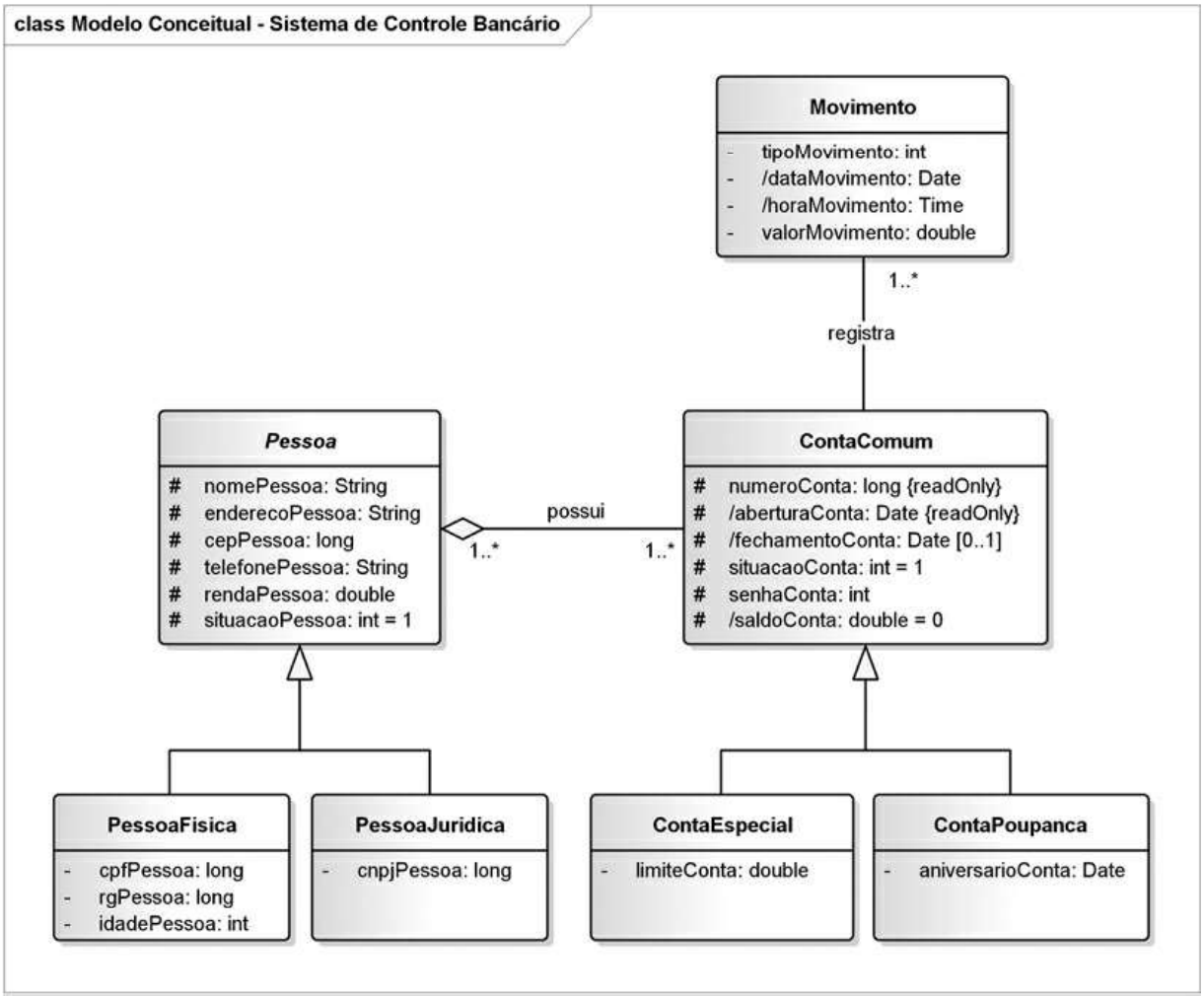

Figura 4.50 – Diagrama de Classes para o Sistema de Controle Bancário – Modelo Conceitual.

Neste modelo, foram identificadas as seguintes classes:

- `ContaComum` – Essa classe foi parcialmente descrita nas seções anteriores. Seu objetivo é armazenar as contas-correntes comuns, sem limite de cheque especial, gerenciadas pelo banco. Nesse tipo de conta, não é possível realizar saques além do valor real depositado. A classe `ContaComum` está associada a duas especializações ou subclasses que herdam seus atributos e métodos, representadas pelas classes `ContaEspecial` e `ContaPoupanca`.

A classe `ContaComum` tem os atributos número da conta (`numeroConta`) do tipo `long`, data da abertura (`aberturaConta`) e data de encerramento (`fechamentoConta`) do tipo `Date`, `situacaoConta` (se está ativa ou inativa), `senhaConta` do tipo `int` e `saldoConta` do tipo `double`. Observe que os atributos da classe `ContaComum` são todos protegidos, o que os torna visíveis às suas subclasses. Os atributos dessa classe têm ainda características extras, como a definição de que `aberturaConta`, `fechamentoConta` e `saldoConta` são atributos que receberão algum tipo de cálculo (ou atribuição), que o atributo `fechamentoConta` não necessariamente precisará ser informado e que os valores iniciais de `situacaoConta` e `saldoConta` serão, respectivamente, 1 (ao abrir uma conta, ela se tornará ativa) e 0 (enquanto nenhum depósito for realizado).

É importante destacar que o tipo `Date` refere-se a uma classe e se esta não for implementada pela linguagem adotada para a codificação do sistema, deverá ser criada.

Nas seções anteriores, apresentamos exemplos de restrições que poderiam ser aplicadas aos atributos dessa classe, mas com o objetivo de não deixar o diagrama muito poluído, com grande quantidade de informações, preferimos detalhar esse tipo de informação em diagramas separados, quando necessário. Poderíamos acrescentar, ainda, as restrições `readOnly` aos atributos `numeroConta` e `aberturaConta`, uma vez que, após a criação, o número da conta e sua data de abertura não podem ser modificados.

- `ContaEspecial` – A classe `ContaEspecial` representa as contas que permitem ao correntista sacar valores superiores a seu saldo, até um limite estabelecido. Essa classe tem um atributo particular, além dos herdados da classe `ContaComum`, chamado `limiteConta` do tipo `double`, que representa o limite máximo que o correntista pode retirar além do saldo de sua conta.
- `ContaPoupanca` – A classe `ContaPoupanca`, como o próprio nome indica, representa as contas de poupança mantidas pela instituição bancária. Essa classe tem um único atributo particular, além dos herdados, chamado `aniversarioConta` do tipo `Date`, que representa a

data em que o valor depositado na conta renderá juros, que não é necessariamente equivalente à data de abertura, uma vez que as contas de poupança não exigem um depósito quando de sua abertura.

- `Pessoa` – A classe `Pessoa` armazena as informações gerais dos clientes. Tanto pessoas físicas como jurídicas podem possuir contas na instituição bancária. A classe `Pessoa` tem uma associação do tipo agregação com a classe `ContaComum`, o que significa que uma ou mais instâncias dessa classe ou de suas subclasses complementam as informações armazenadas por instâncias da classe `Pessoa`. Assim, sempre que uma determinada pessoa for consultada, as informações sobre suas contas deverão ser também apresentadas.

Observe que a multiplicidade da associação **possui** existente entre as classes `Pessoa` e `ContaComum` é muitos (*) em ambas as extremidades, significando que uma pessoa pode possuir, no mínimo, uma e, no máximo, muitas contas e uma conta pode ser possuída por uma ou mais pessoas, como no caso de contas conjuntas. O leitor pode se perguntar se não haveria necessidade de inserir uma classe associativa ou intermediária entre essa associação, porém isso somente seria necessário se houvesse um atributo específico relativo a uma determinada pessoa possuindo uma determinada conta que não pudesse ser armazenado nem na classe `Pessoa` nem na classe `ContaComum`.

É preciso notar que a classe `Pessoa` é uma superclasse abstrata (por isso seu nome está em itálico), que não tem instâncias e, portanto, não interage realmente com a classe `ContaComum` nem com suas especializações. As classes que interagem com a classe `ContaComum` são as subclasses `PessoaFisica` e `PessoaJuridica`, que herdam todos os atributos da classe `Pessoa` e ainda têm seus próprios atributos particulares.

Os atributos da classe `Pessoa` são nome, endereço e telefone do tipo `String`, CEP do tipo `long`, renda do tipo `double` e situação (se está ativa – possui contas ainda não encerradas – ou inativa) do tipo `int`.

- `PessoaFisica` – A classe `PessoaFisica` é uma subclasse derivada da classe `Pessoa`, representando as pessoas físicas que possuem ou possuíram contas ativas na instituição bancária. Essa classe herda todos

os atributos de sua superclasse, tendo os atributos extras CPF e carteira de identidade (RG) do tipo `long` e idade do tipo `int`.

- `PessoaJuridica` – Essa classe também é uma subclasse derivada da classe `Pessoa`, representando as pessoas jurídicas que possuem ou possuíram contas ativas na instituição. Da mesma maneira, a classe `PessoaJuridica` herda todos os atributos da classe `Pessoa`, possuindo somente o atributo particular CNPJ da empresa do tipo `long`.
- `Movimento` – Essa classe é responsável por armazenar as transações ocorridas nas contas, ou seja, todos os saques e depósitos. A classe `Movimento` tem como atributos o tipo de movimento (convencionamos atribuir 0 para saque e 1 para depósito) do tipo `int`, a data do movimento (`dataMovimento`) do tipo `Date`, a hora do movimento (`horaMovimento`) do tipo `Time` e o valor movimentado do tipo `double`. Observe que o tipo `Time`, da mesma forma que o `Date`, refere-se a classes e não a tipos primitivos, portanto se essas classes não forem implementadas pela linguagem, terão que ser igualmente codificadas. A classe `Movimento` tem uma associação binária simples com a classe `ContaComum`. A multiplicidade dessa associação demonstra que uma conta terá, no mínimo, um movimento (após a abertura, é obrigatório depositar algum valor) e, no máximo, muitos. Essa associação não caracteriza uma composição porque as informações da classe `Movimento` não complementam as informações da classe `Conta`, quando esta for consultada.

4.8 Como Identificar Classes

O leitor pode se perguntar como essas classes foram identificadas e de que maneira se concluiu que seriam as classes corretas para constar no modelo conceitual. Isso é um pouco influenciado pela própria experiência da equipe, todavia existem algumas técnicas para identificar as "classes candidatas", ou seja, as classes que têm probabilidade de compor o modelo. A estratégia mais comum é examinar a descrição dos requisitos, podendo estas estar contidas em documentos de requisitos ou na documentação dos casos de uso, por exemplo.

Nessa análise textual, costuma-se procurar por substantivos e verbos (ou

descrições de ações). Os substantivos podem representar as classes candidatas ou seus atributos (é necessário verificar os possíveis sinônimos de um substantivo e agrupá-los em uma classe ou atributo único), enquanto os verbos podem identificar as operações válidas para uma determinada classe ou associações entre as classes.

Também é útil procurar descrições de restrições ou condições nos substantivos e verbos que foram identificados anteriormente. Os requisitos não funcionais costumam conter esse tipo de informação e são úteis para identificar as possíveis restrições que deveriam ser aplicadas às classes, seus atributos, operações e associações.

Dessa maneira, para cada substantivo identificado na documentação de requisitos, deve-se tentar representar uma classe correspondente no modelo conceitual ou, eventualmente, um atributo contido em uma das classes do modelo. Em seguida, para cada descrição de ação, deve-se procurar identificar um comportamento ou combinação de comportamentos associado a uma classe. Verbos normalmente identificam ações, contudo, eventualmente, podem identificar uma associação entre as classes. É necessário certificar-se também de que as operações (comportamentos) de cada classes recebam os dados corretos para que o comportamento seja executado a contento. Em geral, esses dados serão atributos que deverão estar contidos em uma classe. É importante destacar que esse processo pode (e deve) passar por refinamentos, em que o modelo inicial deve ser analisado, corrigido (se necessário) e melhorado.

Assim, se analisarmos o item 3.15 no capítulo 3, poderemos identificar alguns substantivos, como clientes, pessoas físicas, pessoas jurídicas, contas comuns, contas especiais, contas de poupança e movimentos.

Para desenvolver o modelo conceitual apresentado na figura 4.50, consideramos que os termos “clientes” e “pessoas” são sinônimos e preferimos adotar a terminologia genérica “Pessoa” para identificar uma classe, enquanto Pessoa Física e Pessoa Jurídica tornaram-se especializações dessa classe. Cada um dos tipos de conta (comum, especial e poupança) foi identificado como uma classe, porém as duas últimas foram representadas como especializações da classe `ContaComum`. Também **Movimento** tornou-se uma classe e foi associada à classe `ContaComum` e, automaticamente, também às suas especializações, por meio da associação

de herança.

Nesse modelo, como dito anteriormente, não são identificados métodos, porque estes fazem parte da solução e só são representados na fase de projeto. Assim, os verbos não foram analisados com muita profundidade nessa fase e apenas serviram para auxiliar a identificar as associações "**registra**" entre as classes **ContaComum** e **Movimento** e "**possui**" entre as classes **Pessoa** e **ContaComum**. Os verbos foram melhor analisados durante a fase de projeto, na qual foi produzido o modelo de domínio apresentado a seguir.

Obviamente, o modelo explicado nesta seção já passou por alguns refinamentos e o diagrama apresentado é a versão final do modelo conceitual para este estudo de caso.

4.9 Exemplo de Modelo de Domínio

A seguir, apresentaremos o modelo de domínio desse sistema, no qual detalhamos os métodos necessários às classes identificadas no modelo conceitual. Lembramos que o modelo de domínio costuma ser produzido somente na fase de projeto e seus métodos são identificados por meio da modelagem de diagramas de interação correspondentes aos processos representados no modelo de casos de uso.

A figura 4.51 apresenta o mesmo diagrama de classes para o sistema de controle bancário explicado anteriormente, mas, desta vez, o diagrama representa o modelo de domínio e enfoca a solução do problema a ser resolvido pelo sistema. Como podemos observar, foram acrescentados métodos e navegabilidade a esse modelo.

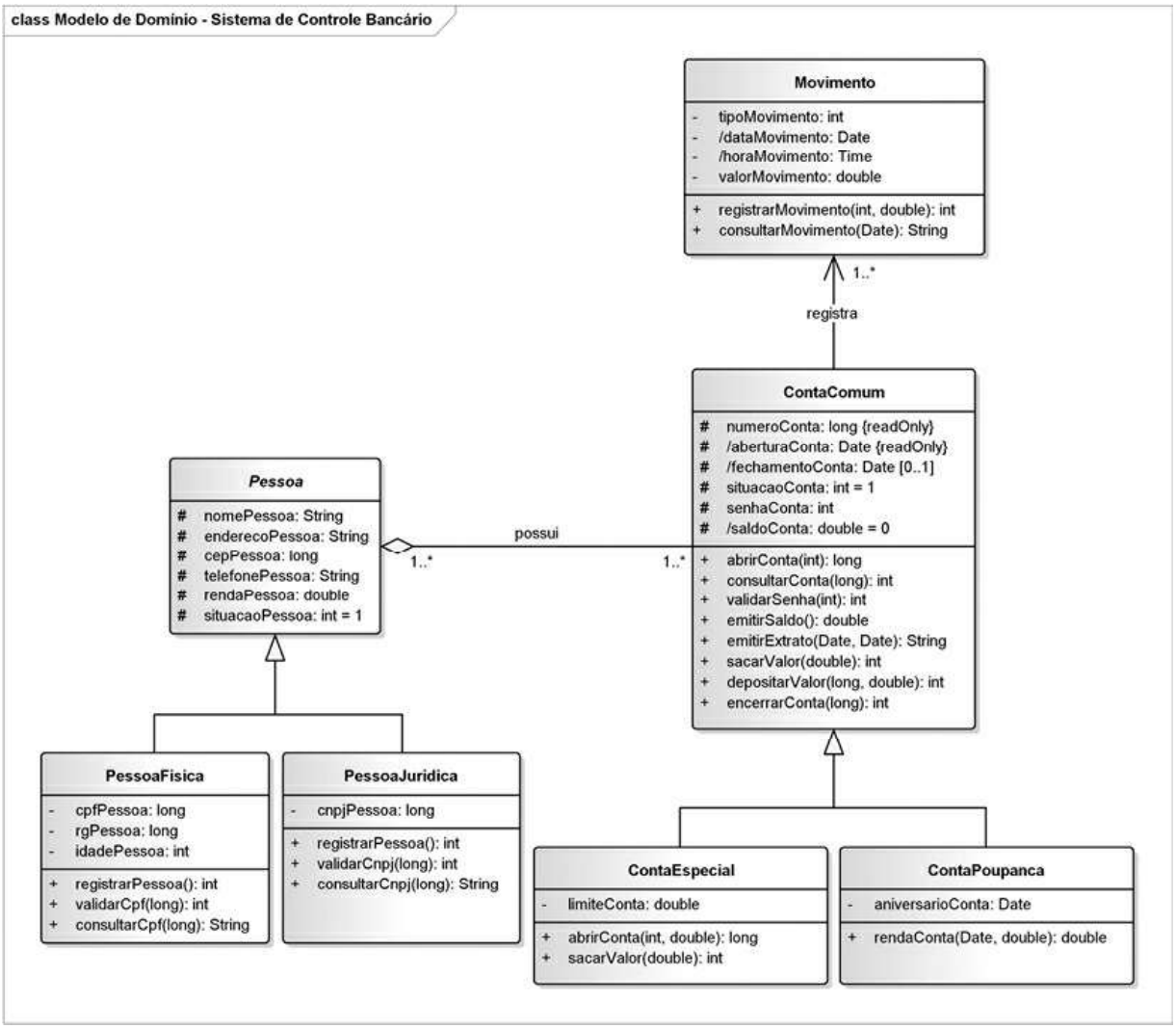

Figura 4.51 – Diagrama de Classes para o Sistema de Controle Bancário – Modelo de Domínio.

É importante destacar que todas as outras classes do diagrama, com exceção da classe `Pessoa` (que é uma classe abstrata, como demonstra seu nome em itálico), são classes de entidade, mas optamos por não utilizar o estereótipo **<<entity>>** porque este, por ser gráfico, modifica o desenho-padrão da classe e esconde seus atributos e métodos, o que atrapalharia a explicação desse diagrama. A seguir, detalharemos os métodos identificados para cada classe, que foram descobertos quando do detalhamento dos casos de uso por meio de diagramas de sequência e comunicação que serão explanados nos próximos capítulos.

- `ContaComum` – Os métodos identificados para essa classe foram:

Método	Descrição
abrirConta	A função desse método é abrir uma nova conta, onde um novo objeto dessa classe será instanciado, de forma que esse método agirá quase como um método construtor. Esse método recebe como parâmetro somente a senha da conta, a ser definida pelo cliente, uma vez que o número da conta é gerado automaticamente e retornado pelo método. A data de abertura é tomada do sistema, a de encerramento é deixada indefinida e a situação e o saldo têm valores iniciais predefinidos. O retorno do método é o próprio número da conta, se o método for concluído com sucesso, ou zero, se ocorrer algum erro durante sua execução.
consultarConta	Tem por objetivo descobrir se uma determinada conta existe. Recebe um `long` que armazena o número da conta e retorna um `inteiro` para determinar se a conta foi encontrada (1) ou não (0).
validarSenha	Determina se a senha informada é válida. O método recebe a senha como parâmetro e retorna um inteiro determinando se a senha é a correta (1) ou não (0).
emitirSaldo	Retorna o valor contido na conta, retornando um valor `double` que armazena o saldo da conta. Como esse método só pode ser chamado após o método `Consulta`, não é preciso informar o número da conta, visto que já foi informada.
emitirExtrato	Retorna os movimentos realizados na conta em um determinado período. Recebe como parâmetro as datas iniciais e finais do extrato. O método tem a necessidade de solicitar a execução de outro método sobre a classe `Movimento`.
sacarValor	Diminui o valor solicitado para saque do saldo da conta. Recebe como parâmetro o valor a ser retirado do saldo e retorna verdadeiro, se foi possível realizar a operação, ou falso, em caso contrário. O retorno falso pode ocorrer por a conta não ter saldo suficiente para o valor solicitado. Como o método `sacarValor` só poderá ser chamado após a conta ter sido consultada e sua senha, validada, não é necessário receber o número da conta, pois este já foi informado no momento da consulta da conta.
depositarValor	Soma o valor fornecido pelo cliente ao saldo de uma conta. Recebe como parâmetro o número da conta (o depósito pode ser relativo à outra conta) e o valor a ser depositado. Tanto esse método como o `sacarValor` disparam um método na classe `Movimento` para registrar o movimento realizado sobre a conta.
encerrarConta	Encerra uma conta já existente, tornando-a inativa. Não é um método destrutor, pois apenas altera o valor do atributo `situacao` para 0, indicando que a conta se encontra inativa. Possivelmente, faz o mesmo com o cliente se esta for a única conta por ele possuída. Uma conta não pode ser excluída: é preciso mantê-la para fins de histórico bancário. Esse método não recebe parâmetros, já que para ser encerrada a conta precisa ser, primeiro, consultada por meio do método `consultarConta` e, depois

disso, o número da conta já estará armazenado na memória. Esse método retorna 1, caso a conta tenha sido encerrada com sucesso, ou 0, caso não tenha sido possível encerrá-la.

O leitor talvez se pergunte o porquê de o método `abrirConta` ser considerado um método construtor. Na verdade, recomenda-se que os métodos construtores e destrutores não sejam representados no diagrama de classes porque a forma como são implementados varia muito entre as linguagens de programação. Basicamente, um método criador aloca memória para uma instância de uma determinada classe e, depois, determina valores para seus atributos, enquanto um método destrutor libera a memória utilizada por uma determinada instância. Ao seguir esse raciocínio, o método `abrirConta` pode perfeitamente implementar a rotina para alocar memória para uma nova instância da classe `ContaComum` ou chamar um método para isso antes de finalizar a abertura da conta. Na verdade, esse método não seria totalmente obrigatório, já que a própria classe controladora poderia chamar um método construtor, mas optamos por defini-lo por considerarmos esse comportamento importante para o sistema.

- `ContaEspecial` – A classe `ContaEspecial` herda todos os métodos da classe `ContaComum`, mas tem ainda três métodos particulares, a saber:

Método	Descrição
`abrirConta`	Este é uma redeclaração do método `abrirConta` da classe `ContaComum`. O método precisa ser redeclarado pela necessidade de se incluir o atributo `limite`, inexistente na classe `ContaComum`, no momento da abertura da conta. Essa redeclaração do método `abrirConta` caracteriza um exemplo de polimorfismo, já que a classe redeclarada tem o mesmo nome, mas algumas diferenças em sua implementação.
`sacarValor`	O segundo método da classe `ContaEspecial` também é uma redeclaração do mesmo método existente na classe `ContaComum`. Esse método precisa ser redeclarado para incluir o uso do atributo `limite`, pois um correntista que possua uma conta especial pode sacar valores superiores a seu saldo real até o limite estabelecido pelo atributo de mesmo nome. Tal redeclaração caracteriza outro exemplo de polimorfismo, uma vez que a chamada do método permanece a mesma, diferenciando-se na forma como o método é implementado.

- `ContaPoupanca` – Da mesma forma que a classe `ContaEspecial`, essa classe herda todos os métodos da classe `ContaComum`, tendo somente um

método extra:

Método	Descrição
`rendaConta`	Recebe como parâmetros a data atual e o percentual de juros que as contas com aniversário no dia receberão. O método aplica-se a todas as instâncias cujo dia da data de aniversário seja igual ao dia da data atual.

- `PessoaFisica` – Essa classe tem os seguintes métodos:

Método	Descrição
`registrarPessoa`	É responsável por instanciar um novo objeto da classe. Uma vez que esse método recebe como parâmetro todos os atributos da classe, com exceção da situação, cujo valor inicial é 1 (ativo), optamos por não detalhar a assinatura do método para não deixar a classe larga demais, o que atrapalharia a visualização da figura.
`validarCPF`	É utilizado para determinar se o CPF informado é válido. Esse método é chamado quando do registro de uma nova pessoa física e recebe como parâmetro um `long` que armazenará o valor do CPF a validar.
`consultarCpf`	Permite consultar uma pessoa por seu CPF. Se o CPF for encontrado, retornará uma String com os dados do cliente.

- `PessoaJuridica` – Os métodos dessa classe são descritos a seguir:

Método	Descrição
`registrarPessoa`	Tem a mesma função do método de mesmo nome explicado na classe anterior, diferenciando-se por não registrar o CPF nem o RG da pessoa, e sim o CNPJ da empresa.
`validarCnpj`	É utilizado para determinar se o CNPJ informado é válido. Esse método é chamado quando do registro de uma nova pessoa jurídica e recebe como parâmetro um `long` que armazenará o valor do CNPJ a validar.
`consultarCnpj`	Permite consultar uma pessoa jurídica por seu CNPJ. Se o CNPJ for encontrado, retornará uma String contendo os dados do cliente.

- `Movimento` – Essa classe tem os métodos explanados a seguir:

Método	Descrição
`registrarMovimento`	É responsável por registrar cada movimento ocorrido em alguma conta. Esse método recebe como parâmetros o tipo de movimento (se foi saque ou depósito) do tipo `int` e o valor do movimento do tipo `double`, pois a data e a hora do movimento são capturadas diretamente do sistema. O método retornará 1 se conseguir registrar o movimento, caso contrário, 0. Esse método é chamado pelos métodos `sacarValor` e `depositarValor` declarados na classe `ContaComum`.
`consultarMovimento`	É utilizado para consultar todos os movimentos pertencentes a uma determinada data, recebe como parâmetro a data da consulta e retorna uma `string` contendo os valores dos atributos de cada

objeto.

Aqui, é necessário fazer uma observação principalmente em relação aos exemplos seguintes. A rigor, uma classe deve ter um `get` e um `set` para cada um de seus atributos. No entanto, torna-se inviável e enfadonho criar esses métodos para cada classe; se a classe tiver muitos atributos, tornar-se-á muito grande e, por conseguinte, ficará difícil ler a representação do diagrama como um todo. Por esse motivo, recomenda-se não representar esse tipo de método em diagramas de classe, mesmo porque a informação que esse tipo de método acrescenta é pífia. Assim, optamos por utilizar métodos genéricos como registrar ou consultar, para instanciar ou retornar todos os valores de uma classe com o objetivo de simplificar o modelo, o que não impede que o leitor proceda de forma diferente, se assim achar necessário.

Também é possível criar diagramas referentes a um caso de uso específico, contendo somente as classes de fronteira, controle e entidade envolvidas no processo representado pelo caso de uso. No estudo de caso do sistema de controle bancário existem poucos exemplos que possam ser aplicados nesse sentido, uma vez que a maioria dos processos envolve praticamente todas as classes, no entanto podemos ilustrar essa prática por meio do processo emitir saldo, que não envolve objetos da classe `Movimento`, conforme demonstra a figura 4.52.

No exemplo apresentado na figura 4.52, ilustramos as classes envolvidas no processo para emitir o saldo de uma conta. Pode-se perceber que a classe `VisaoEmitirSaldo` é a classe de fronteira que representa a interface com o usuário (pode haver mais de uma interface para o mesmo processo para englobar diversas dispositivos e formas de acesso para o mesmo processo) e a classe `ControleEmitirSaldo` (ou melhor, suas instâncias) é responsável por controlar o processo para emissão do saldo de uma conta. As classes de entidade aqui presentes foram explicadas anteriormente.

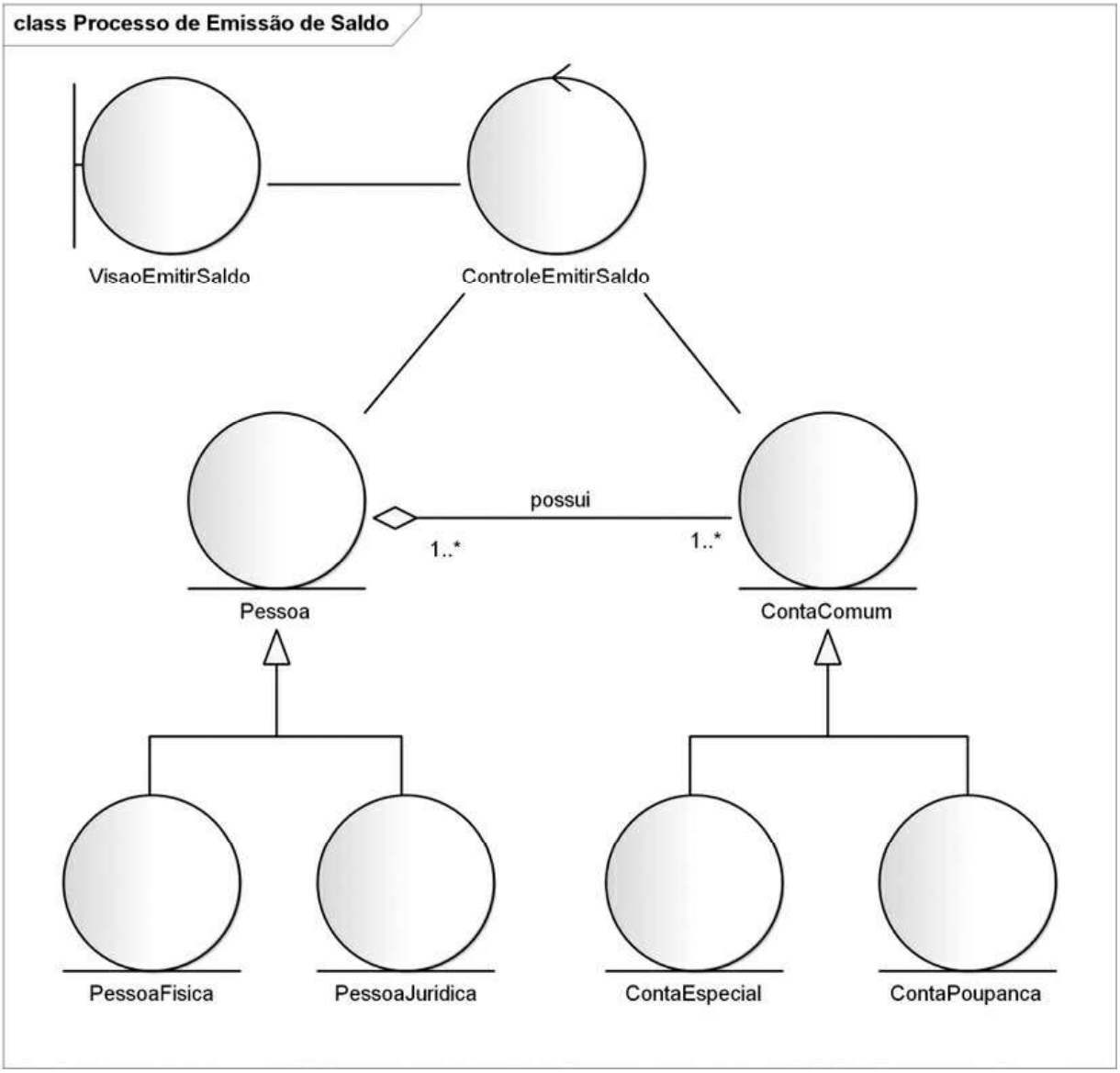

Figura 4.52 – Diagrama de Classes para o Processo de Emissão de Saldo – Sistema de Controle Bancário.

Nos modelos seguintes, apresentaremos somente modelos de domínio contendo classes de entidade, ou seja, as classes contidas na camada de modelo do padrão MVC. Consideraremos que as classes de fronteira e controle estão definidas em suas respectivas camadas.

4.10 Exemplo de Diagrama de Classes – Sistema de Telefone Celular

Nesta seção, apresentaremos o diagrama de classes referente ao modelo de domínio para o sistema de telefone celular, cuja modelagem foi iniciada no