A linguagem de modelagem unificada UML – Unified Modeling Language, é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos.

É uma linguagem de modelagem de propóristo geral que pode ser aplicada a todos os domínios de aplicação. Não é uma linguagem de programação, é uma linguagem de modelagem.

Objetivo é auxiliar os engenheiros de software a definirem as características do sistema, como seus requisitos, seu comportamento, sua lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sore o qual o sistema deverá ser implantado.

UML 2 – **Orientação a Objetos**

**Classificação, Abstração e Instanciação**: fazemos a classificação de objetos a partir de um padrão, ele anda, tem quatro rodas e buzina, pode ser classificado como um carro. Ao pensar nas características desse carro e seus métodos de funcionamento estamos fazendo a abstração. A instância é nada mais que a dependência do o objeto diretamente a classe, então ao instanciar um objeto estamos declarando que ele faz parte de determinada classe.

**Classes e Objetos:** uma classe representa uma categoria e os objetos são os membros ou exemplos dessa categoria.

Na UML, uma classe é representada por um retângulo que pode ter até três divisões: Nome (a única obrigatória), possíveis Atributos pertencentes a classe, e possíveis Métodos. Em geral uma classe tem atributos e métodos, mas é possível encontrar classes que contenham apenas uma dessas características ou mesmo nenhuma delas, como no caso de classes abstratas.

**Atributos ou Propriedades:** representam as características de uma classe, ou seja, as peculiariadades que costumam variar de um objeto para outro, como nome, CPF e idade em um objeto da classe Pessoa ou a placa e a cor de um objeto da classe Carro; permite diferenciar um objeto de outro da mesma classe em razão de tais variações.

Na UML, os atributos são apresentados na segunda divisão da classe e contêm, normalmente, duas informações: o nome que identifica o atributo e o tipo de dado que o atributo armazena, como **integer, float ou string**. Na realidade, não é exatamente a classe que contém os atributos, mas, sim, as instâncias, os objetos dessa classe. Nunca poderemos trabalhar com a classe Pessoa propriamente dita, apenas com suas instâncias, como João, Pedro ou Paulo, que são nomes que identificam três objetos da classe Pessoa.

**Operações, Métodos ou Comportamentos:**  também conhecidos como operações que representa atividade que um objeto de uma classe pode executar. Um método pode receber ou não parâmetros (valores utilizados durante a execução do método) e, em geral, tende a retornar valores. Esses valores podem representar o resultado da operação executada ou simplesmente indicar se o processo foi concluído com sucesso ou não.

**Visibilidade:** é utilizada para indicar o nível de acessibilidade de um determinado atributo ou método, sendo representáda à esquerda destes. Existem basicamente quatro modos de visibilidade: público, protegido, privado e pacote.

• A visibilidade privada é representada por um símbolo de menos (-) e significa que somente os objetos da classe detentora do atributo ou método poderão enxergá-lo. Somente as instâncias da classe Pessoa propriamente ditas podem enxergar esses atributos.

• A visibilidade protegida é representada pelo símbolo de sustenido (#) e determina que, além dos objetos da classe detentora do atributo ou método, também os objetos de suas subclasses poderão ter acesso a este.

• A visibilidade pública é representada por um símbolo de mais (+) e determina que o atributo ou método pode ser utilizado por qualquer objeto. Objetos de outras classes enxerguem o método.

• A visibilidade pacote é representada por um símbolo de til (~) e determina que o atributo ou método é visível por qualquer objeto dentro do pacote. Somente elementos que fazem parte de um pacote podem ter essa visibilidade. Nenhum elemento fora do pacote poderá ter acesso a um atributo ou método com essa visibilidade.

É importante destacar que normalmente os atributos costumam ser privados ou protegidos, enquanto os métodos costumam ser públicos. A declaração de atributos como privados, ou eventualmente protegidos, é altamente recomendada, pois isto garante o encapsulamento dos atributos. Um atributo privado, além de só ser visível por objetos de sua classe, só poderá ser acessado por meio de métodos. Assim, objetos de outras classes não terão conhecimento sobre quais atributos estão contidos na classe em questão e nem poderão acessá-los. Eles saberão da existência dos métodos da classe (quando forem públicos), mas não como o método manipula seus atributos.

**Herança:**  é uma das características mais poderosas e importantes da orientação a objetos, pois permite o reaproveitamento de atributos e métodos, otimizando o tempo de desenvolvimento, além de permitir a diminuição de linhas de código, como como facilitar futuras manutenções.

A herança na orientação a objetos trabalha com os conceitos de superclasses e subclasses. Uma superclasse, também chamada de classe-mãe, contém classes derivadas dela, chamadas subclasses, também conhecidas como classes-filha. As subclasses, ao serem derivadas de uma superclasse, herdam suas características, ou seja, seus atributos e métodos.

Além disso, a herança permite trabalhar com especializações. Podemos criar classes gerais, com características compartilhadas por muitas classes, mas que tenham pequenas diferenças entre si. Assim, criamos uma classe geral com as características comuns a todas as classes e diversas subclasses a partir dela, detalhando apenas os atributos ou métodos exclusivos destas. Uma subclasse pode se tornar uma superclasse a qualquer momento, bastando para tanto que se derive uma subclasse dela.

**Herança Múltiplas:** ocorre quando uma subclasse herda características de duas ou mais superclasses. No caso, uma subclasse pode herdar atributos e métodos de diversas superclasses. No entanto, não são todas as linguagens de programação orientadas a objetos que fornecem esse tipo de recurso.

**Polimorfismo:** o conceito de polimorfismo está associado à herança. O polimorfismo trabalha com a redeclaração de métodos previamente herdados por uma classe. Esses métodos, embora semelhantes, diferem de alguma forma da implementação utilizada na superclasse, sendo necessário, portanto, reimplementá-los na subclasse.

Porém, para evitar ter de modificar o código-fonte, inserindo uma chamada a um método com um nome diferente, redeclara-se o método com o mesmo nome declarado na superclasse. Assim, podem existir dois ou mais métodos com a mesma nomenclatura, diferenciando-se na maneira como foram implementados, sendo o sistema responsável por verificar se a classe da instância em questão contém o método declarado nela própria ou se o herda de uma superclasse. Em uma situação em que existam diversas subclasses que herdem um método de uma superclasse, se uma delas redeclarar esse método, ele só se comportará de maneira diferente nos objetos da classe que o modificou, permanecendo igual à forma como foi implementado na superclasse para os objetos de todas as demais classes. Métodos que são redeclarados em subclasses e apresentam um comportamento diferente do método de mesmo nome contido na superclasse são chamados de métodos polimórficos.

**Diagrama de Casos de Uso**

O diagrama de casos de uso procura possibilitar a compreensão do comportamento externo do sistema (em termos de funcionalidades oferecidas por ele) por qualquer pessoa com algum conhecimento sobre o problema enfocado, tentando apresentar o sistema por intermédio de uma perspectiva dos usuários. Esse diagrama costuma ser utilizado, sobretudo, no início da modelagem do sistema, principalmente nas etapas de elicitação e análise de requisitos, embora venha a ser consultado e possivelmente modificado durante todo o processo de engenharia e sirva de base para a modelagem de outros diagramas.

Esse diagrama tem por objetivo apresentar uma visão externa geral das funcionalidades que o sistema deverá oferecer aos usuários, sem se preocupar em profundidade com a questão de como tais funcionalidades serão implementadas. O diagrama de casos de uso tenta identificar os tipos de usuários que interagirão com o sistema, quais papéis eles assumirão e quais funções um usuário específico poderá requisitar.

Por utilizar uma linguagem menos formal e apresentar uma visão geral do comportamento do sistema a ser desenvolvido, o diagrama de casos de uso pode e deve ser apresentado durante as reuniões iniciais com os stakeholders para ilustrar as funcionalidades e o comportamento do sistema de informação, facilitar a compreensão dos stakeholders do que se pretende desenvolver e auxiliar na identificação de possíveis falhas de especificação, verificando se os requisitos do sistema foram bem compreendidos. É bastante útil e recomendável que um diagrama de casos de uso seja apresentado aos clientes com um protótipo, o que permitirá que um complemente o outro.

**Atores:**  costumam representar os papéis desempenhados pelos diversos usuários que poderão utilizar, de alguma maneira, os serviços e funções do sistema. Eventualmente, um ator pode representar algum hardware especial ou mesmo outro software que interaja com o sistema, como no caso de um sistema integrado, por exemplo.

Os atores mais comuns (pessoas) são representadas pelo tradicional boneco de palito e seu nome/descrição (Gerente, Funcionário, Cliente...), já os atores que representam componentes (hardware ou sistemas/software), são também representados pelo boneco palito, porém em sua descrição é uma convenção utilizar <<>> em sua descrição para distinguir o tipo de ator atribuindo-lhe um estereótipo que o classifica como um ator fora do ‘comum’ (<<system>> Sistema integrado).

Diagrama

Descrição gerada automaticamente

**Como identificar atores?** Procurar identificar as entidades externas que interagirão com o sistema, tanto usuários humanos, como também, eventualmente, softwares e/ou hardwares. Também se deve procurar agrupar usuários com características semelhantes, que utilizarão as mesmas funcionalidades no sistema e possuirão os mesmos níveis de permissão, identificando-os como um ator único.

Algumas perguntas podem ser úteis para identificar candidatos a atores, como:

• Que tipos de usuários poderão utilizar o sistema?

• Quais usuários estão interessados ou utilizarão quais funcionalidades e serviços do software? • Quem fornecerá informações ao sistema?

• Quem utilizará as informações do sistema?

• Quem poderá alterar ou mesmo excluir informações do sistema?

• Existe algum outro software que interagirá com o sistema?

• Existe algum hardware especial (como um robô, por exemplo) que interagirá com o software?

Os candidatos a atores devem ser listados e deve-se tentar atribuir responsabilidades e objetivos a cada um deles, ou seja, metas que cada ator poderia desejar atingir ao utilizar o software. Atores para os quais não é possível atribuir um objetivo dificilmente serão atores verdadeiros e deverão ser eliminados.

**Casos de Uso:** são utilizados para capturar os requisitos funcionais do sistema, ou seja, referem-se a servuços, tarefas ou funcionalidades identificados como necessários ao software e que podem ser utilizados de alguma maneira pelos atores que interagem com o sistema, eles expressam e documentam os comportamentos pretendidos para as funções do software.

São representados por elipses contendo dentro de si um texto que descreve a que funcionalidade o caso de uso se refere. O texto contido em um caso de uso costuma iniciar com um verbo denotando a ação que será realizada quando de sua execução. Ex.: Abrir Conta.

Casos de uso podem ser classificados em casos de uso primários ou secundários. Um caso de uso é considerado primário quando se refere a um processo importante, que enfoca um dos requisitos funcionais do software, como realizar um saque ou emitir um extrato em um sistema de controle bancário. Já um caso de uso secundário se refere a um processo periférico, como a manutenção de um cadastro ou a emissão de um relatório simples. Em situações em que há um grande número de casos de uso, é recomendável limitar a representação de casos de uso secundários, representando-os de maneira geral ou mesmo não os representando de forma alguma, para evitar poluir demais o diagrama.

Os casos de uso costumam ser documentados. A documentação de um caso de uso fornece instruções em linhas gerais de como será seu funcionamento, quais atores poderão utilizá-lo, quais atividades deverão ser executadas pelos atores e pelo sistema, qual evento forçará sua execução e quais suas possíveis restrições, entre outras informações.

Normalmente, as ações contidas na documentação de um caso de uso são descritas em termos gerais, embora nada impeça que o engenheiro de software insira detalhes mais técnicos, até mesmo de implementação na documentação, o que não é recomendado. A documentação deve representar o comportamento de um caso de uso da maneira o mais completa possível, mas sem detalhes técnicos.,

**Documentação de Casos de Uso:** A documentação de um caso de uso costuma descrever, por meio de uma linguagem bastante simples, informações como a função em linhas gerais do caso de uso, quais atores interagem com ele, quais etapas devem ser executadas pelo ator e pelo sistema para que o caso de uso execute sua função, quais parâmetros devem ser fornecidos e quais restrições e validações o caso de uso deve possuir.

Não existe um formato específico de documentação para casos de uso definido pela UML propriamente dita, porém há formatos propostos em diversas literaturas técnicas. O diagrama de casos de uso, todavia, é flexível à forma como se deseja documentá-lo, permitindo que se documente um caso de uso da maneira que se considerar melhor.

Documentação do Caso de Uso Abertura de Conta

|  |  |
| --- | --- |
| Nome do Caso de Uso | UC01 – Abrir Conta |
| Caso de Uso Geral |  |
| Ator Principal | Funcionário |
| Atores Secundários | Cliente |
| Resumo | Esse caso de uso descreve as etapas percorridas por um cliente, intermediado por um funcionário, para abrir uma conta-corrente |
| Pré-condições | O pedido de abertura precisa ter sido previamente aprovado |
| Pós-condições | É necessário realizar um depósito inicial |
| Cenário Principal | |
| Ações do Ator | Ações do Sistema |
| 1 - O funcionário informa o CPF ou CNPJ do cliente e consulta seu registro |  |
|  | 2 - Consultar cliente por seu CPF ou CNPJ |
| 3 - O cliente informa a senha da conta |  |
|  | 4 - Abrir conta |
| 5 - O cliente fornece um valor a ser depositado |  |
|  | 6 - Executar caso de uso “Realizar Depósito” para registrar o depósito do cliente |
|  | 7 - Emitir cartão da conta |
| Restrições/Validações | Para abrir uma conta-corrente, é preciso ser maior de idade |
| O valor mínimo de depósito é R$5,00 |
| O cliente precisa fornecer algum comprovante de residência |
| Cenário Alternativo – Manutenção do Cadastro do Cliente | |
| Ações do Ator | Ações do Sistema |
|  | Executar o Caso de Uso “Gerenciar Clientes”, para registar um novo cliente ou atualizar o cadastro do cliente consultado |
| Cenário de Exceção – Cliente menor de idade | |
| Ações do Ator | Ações do Sistema |
|  | Comunicar ao cliente que ele não tem a idade mínima para possuir uma conta-corrente |
|  | Recusar pedido |

Deve-se fornecer uma descrição (para identificá-lo) para o caso de uso que está sendo documentado, em geral a mesma descrição apresentada externamente no caso de uso. É comum que os casos de uso recebam também um código como UC01, por exemplo.

pode haver casos de uso gerais e casos de uso especializados, que herdam as características dos casos de uso gerais. Assim, no item caso de uso geral, deve-se informar, quando existir, o nome do caso de uso geral a partir do qual o caso de uso atual foi derivado. Nesse exemplo, não existe um caso de uso geral e, por isso, o campo foi deixado em branco (poderia ter sido completamente eliminado nessa situação). Porém, se o caso de uso em questão representasse a abertura de uma conta especial derivada do caso de uso “Abrir Conta Comum”, o campo caso de uso geral armazenaria o texto “Abrir Conta Comum”, indicando que o caso de uso documentado é uma especialização do caso de uso “Abrir Conta Comum”.

Os atores secundários são aqueles que interagem em um nível menor com o caso de uso. Não é obrigatório declarar um ator secundário, visto que pode existir ou não, e pode não ser importante declará-lo, dependendo da importância de sua participação no processo.

O “Cenário Principal” também chamado de fluxo, apresenta as ações que devem normalmente ser realizadas quando o serviço representado pelo caso de uso for solicitado. As ações são divididas em ações realizadas pelo ator que interage com o sistema e em ações executadas pelo próprio sistema, numeradas em uma ordem sequencial.

“Cenários alternativos”, como seu nome indica, podem ser executados ou não, dependendo se uma condição for satisfeita.

Podem existir ainda cenários de exceção que determinam ações que devem ser tomadas em situações em que um cenário principal ou alternativo não pode ser concluído, em razão de alguma regra de negócio ter sido transgredida, por exemplo.

Possíveis “Restrições” e “Validações” do caso de uso, com o objetivo de tornar o processo mais consistente.

Alguns autores inserem também, eventualmente, no final da documentação, requisitos não funcionais que especificam condições de usabilidade, desempenho, confiabilidade ou segurança, por exemplo.

**Como identificar os Casos de Uso?** É necessário deterrminar as funcionalidades necessárias ao software, ou seja, todas as funções e serviços que correspondem aos requisitos funcionais declarados pelos stakeholders como necessários ao sistema.

Verificar a lista dos atores que comporão o sistema e quais os objetivos de cada ator. Esses objetivos muitas vezes corresponderão a uma ou mais funcionalidades. Também pode ser útil identificar os eventos externos que afetarão o software. Tais eventos muitas vezes acarretam a execução de uma funcionalidade ou influenciam de alguma forma seu processamento.

Verificar a lista de requisitos funcionais estabelecidos nas conversações com os stakeholders e se cada um deles possui um (ou até mais) caso de uso especificando seu comportamento e, obviamente, qual ator ou atores pode(m) utilizá-lo.

Uma maneira de determinar se uma funcionalidade é real, ou seja, se realmente corresponde a um caso de uso, é verificar quais ações seriam realizadas quando o caso de uso fosse executado. Se não for possível identificar essas etapas, provavelmente este não será um caso de uso real.

Se o número de passos atribuídos a um caso de uso for muito pequeno, deve-se verificar se esse caso de uso não deveria ser englobado por outro, acrescentando seus passos às etapas dele ou mesmo se tornando um cenário alternativo de outro caso de uso.

**Associações:**  Representam interações ou relacionamentos entre os atores e os casos de uso que fazem parte do diagrama ou os relacionamentos entre os casos de uso e outros casos de uso. Os relacionamentos entre casos de uso recebem nomes especiais, como inclusão, extensão e generalização.

A associação entre um ator e um caso de uso é representada por uma linha ligando o ator ao caso de uso, podendo ocorrer que as extremidades da linha contenham setas, indicando o sentido em que as informações trafegam, ou seja, se estas são fornecidas pelo ator ao caso de uso, se são transmitidas pelo caso de uso ao ator ou ambos (nesse último caso, a linha não tem setas, significando que as informações são transmitidas nas duas direções). As setas também servem para indicar quem inicia a comunicação. Além disso, uma associação pode ter uma descrição própria quando é necessário esclarecer a natureza da informação que está sendo transmitida ou dar um nome a esta, se isso for necessário.

**Generalização/Especialização:** este relacionamento aplica os princípios de herança da orientação a objetos, permitindo que os passos descritos em um caso de uso sejam herdados por outros casos de uso que especializam o caso de uso original chamado geral.

É desnecessário colocar a mesma documentação para todos os casos de uso envolvidos, pois toda a estrutura de um caso de uso generalizado é herdada pelos casos de uso especializados. Além disso, os casos de uso especializados herdam também quaisquer possíveis associações de inclusão ou extensão que o caso de uso geral venha a ter, bem como quaisquer associações com os atores que utilizam o caso de uso geral. A associação de generalização/especialização é representada por uma linha com uma seta mais grossa, que indica qual o caso de uso geral (para o qual a seta aponta) e quais os casos de uso especializados (os que se encontram na outra extremidade da seta, apontando para o caso de uso geral).

Diagrama

Descrição gerada automaticamente

Em situações em que esse tipo de associação é empregado, a documentação dos casos de uso especializados deve conter o item “Caso de Uso Geral”, onde será especificado a partir de qual caso de uso geral eles foram especializados.

O relacionamento de generalização/especialização também pode ser aplicado sobre atores, o que é até mais comum. Um ator geral chamado **Pessoa** e dois atores especializados, **Pessoa Física** e **Pessoa Jurídica.**

O relacionamento de generalização/especialização aplicado a atores pode representar níveis de usuários e aproveitar associações já definidas, evitando deixar o diagrama com muitas linhas entrecruzadas.

Diagrama

Descrição gerada automaticamenteDiagrama

Descrição gerada automaticamente

**Include:** A associação de inclusão costuma ser utilizada quando existe um cenário, situação ou rotina comum a mais de um caso de uso. Quando isso ocorre, a documentação dessa rotina é colocada em um caso de uso específico para que outros casos de uso utilizem esse serviço, evitando-se descrever uma mesma sequência de passos em vários casos de uso. **Os relacionamentos de inclusão indicam uma obrigatoriedade, ou seja, quando um determinado caso de uso tem um relacionamento de inclusão com outro, a execução do primeiro obriga também a execução do segundo.** Um relacionamento de inclusão pode ser comparado à chamada de uma sub-rotina ou função, artifício bastante utilizado na maioria das linguagens de programação.

Uma associação de inclusão é representada por uma linha tracejada contendo uma seta em uma de suas extremidades, a qual aponta para o caso de uso incluído no caso de uso posicionado na outra extremidade da linha. As associações de inclusão costumam apresentar também um estereótipo que contém o texto <<include>>.

Diagrama

Descrição gerada automaticamenteDiagrama

Descrição gerada automaticamente

**Extensão:** associações de extensão são utilizadas para descrever cenários opcionais que podem ser estendidos pelos comportamentos de outros casos de uso. Os casos de uso estendidos descrevem cenários que apenas serão executados em situações específicas quando determinadas condições forem satisfeitas. Dessa forma, as associações de extensão indicam a necessidade de um teste no comportamento de um caso de uso para determinar se é necessário executar também o comportamento (os passos) de um caso de uso estendido.

Relacionamentos de extensão representam situações que não ocorrem sempre, o que não significa que sejam incomuns. As associações de extensão têm uma representação muito semelhante às de inclusão, sendo também representadas por uma linha tracejada, diferenciando-se pelo fato de a seta apontar para o caso de uso que utiliza o caso de uso estendido e por haver um estereótipo contendo o texto “extend” em vez de “include”.

Diagrama

Descrição gerada automaticamente

Os processos de Login e Autorregistrar são representados como casos de uso e há uma associação de extensão entre eles, demonstrando que o caso de uso Autorregistrar poderá ser chamado a partir do caso de uso Realizar Login.

Diagrama

Descrição gerada automaticamente

Nesse exemplo, um cliente dirige-se a um funcionário do banco e solicita o encerramento de uma determinada conta. Como sabemos, para encerrar uma conta, é necessário que seu saldo seja igual a zero. O caso de uso Encerrar Conta representa a funcionalidade de encerramento de conta utilizada pelo funcionário do banco e pode, eventualmente, fazer uma chamada ao caso de uso Realizar Saque – se o saldo da conta estiver positivo – ou ao caso de uso Realizar Depósito – se o saldo da conta estiver negativo. Essas associações entre o caso de uso Encerrar Conta e os casos de uso Realizar Saque e Realizar Depósito são associações de extensão porque os casos de uso somente serão chamados pelo caso de uso Encerrar Conta se as condições a eles associadas forem verdadeiras e, nessa situação, mesmo quando forem chamados, será somente um deles.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pag. 133 até 151 - 4.2.6 Generalização/Especialização

**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. Pode ser representado apenas como um modelo conceitual, apresentando as classes, seus atributos e relacionamentos entre elas e também, pode ser representado como modelo de domínio, com uma abordagem mais completa, apresentando métodos para as classes e as especificações de seus atributos.

Na UML uma classe é representada por um retângulo com até três divisões:

1. Nome ou descrição;
2. Atributos e tipos de dados;
3. Métodos.

Não é obrigatório apresentar todas as três divisões em uma classe, pode ter menos ou até mesmo uma, contendo a descrição da classe (obrigatória). Recomenda-se apresentar apenas atributos relevantes para evitar poluição no diagrama.

Uma imagem contendo Diagrama

Descrição gerada automaticamente

**Atributos e Métodos**

A visibilidade é utilizada para indicar o nível de acesso de um determinado atributo ou método, sendo representada à esquerda destes. Existem 4 modos de visibilidade:

A visibilidade privada é representada por um símbolo de menos (-) e significa que somente os objetos da classe detentora do atributo ou métodos poderão enxergá-lo;

A visibilidade protegida é representada pelo símbolo de sustenido (#) e determina que, além dos objetos da classe detentora do atributo ou método, também os objetos de suas subclasses poderão ter acesso a este.

A visibilidade pública é representada por um símbolo de mais (+) e determina que o atributo ou método pode ser utilizado por qualquer objeto.

A visibilidade pacote é representada por um símbolo de til (~) e determina que o atributo ou método é visível por qualquer objeto dentro do pacote. Somente elementos que fazem parte de um pacote podem ter essa visibilidade. Nenhum elemento fora do pacote poderá ter acesso a um atributo ou método com essa visibilidade.

A lista de argumentos (parâmetros) de um método com seu valor de retorno é chamada de assinatura da operação. Esse detalhamento é opcional.

Uma imagem contendo Tabela

Descrição gerada automaticamente

Os atributos de uma classe podem ainda ter características extras, como:

valores iniciais, iniciam o valor do atributo de toda nova instância

* situacaoConta: int = 1

multiplicidade, determina um intervalo de condição de existência

* /fechamentoConta: Date [0..1]

aributo derivado, ou seja, se seus valores são produzidos por meio de algum tipo de cálculo. Tem uma / antes dos seus nomes.

* /saldoConta: double = 0

Uma imagem contendo Tabela

Descrição gerada automaticamente

**Relacionamentos ou Associações**

Uma associação descreve um vínculo que ocorre normalmente entre os objetos de uma ou mais classes. Permitem que as classes compartilhem informações entre si e colaborem para a execução dos processos esecutados pelo sistema.

As associações são representadas por linhas ligando as classes envolvidas, podem ter nome ou título.

**Associação Unária ou Reflexiva**

Ocorre quando existe um relacionamento de um objeto de uma classe com objetos da mesma classe. Por exemplo a classe Funcionário que possui um relacionamento reflexivo chamado Chefia, um objeto gerente pode ser chefe de um objeto operador de máquina. Essa associação determina se um funcionário pode ou não chefiar outros funcionários.

Diagrama

Descrição gerada automaticamente

O valor “0..\*” é conhecida como multiplicidade, a qual procura determinar o número mínimo e o máximo de objetos envolvidos em cada extremidade da associação, além de permitir especificar o nível de dependência de um objeto para com os outros envolvidos na associação. No exemplo apresentado acima, a multiplicidade mostra que um funcionário pode não chefiar ninguém ou pode chefiar um ou mais funcionários. Significa que um funcionário pode ou não chefiar outros funcionários, mas um funcionário tem um e apenas um funcionário como chefe imediato. Quando se há apenas multiplicade em apenas uma extremidade da associação indica que a multiplicidade é de 1 para 1 ou 1..1.

Texto, Carta

Descrição gerada automaticamente

Além do nome/título e a multiplicidade da associação podemos definir o papel de atores envolvidos.

Diagrama

Descrição gerada automaticamente

No exemplo acima, fica evidente que cada objeto na associação desempenha um papel específico: aquele com multiplicidade 1 é o chefe, enquanto os objetos com multiplicidade muitos são os subordinados. Esses papéis podem ter diferentes níveis de visibilidade, podendo ser públicos, protegidos ou privados. Embora o uso de papéis possa facilitar a compreensão da associação, seu excesso pode tornar os diagramas visualmente confusos.

**A classe Funcionario é considerada persistente, o que significa que suas instâncias devem ser preservadas em disco. Não é obrigatório definir atributos de chave primária ou estrangeira como em um modelo entidade-relacionamento. Normalmente, cada objeto da classe já possui um código interno implícito ao ser instanciado, eliminando a necessidade de declarar um atributo exclusivamente para isso.**

**Associação Binária**