Modelagem de classes

Prof. Marcelo Werneck

Engenharia de Software

Introdução

- O modelo de casos de uso fornece uma perspectiva do sistema a partir de um ponto de vista ***externo***.
- De posse da visão de casos de uso, os desenvolvedores precisam prosseguir no desenvolvimento do sistema.

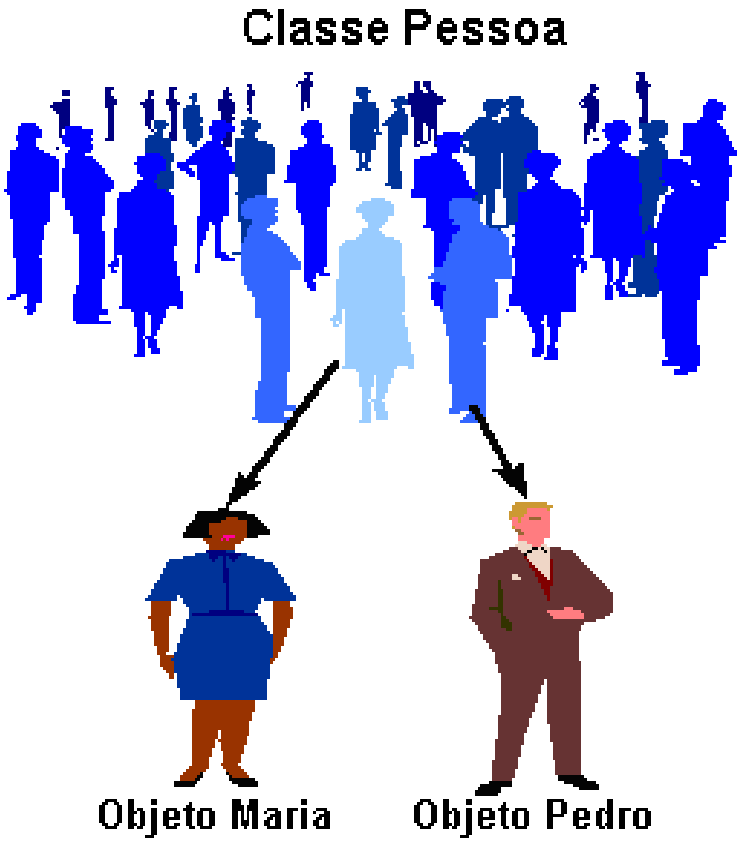
Paradigma Orientado a Objetos

- Útil no desenvolvimento de grandes sistemas
- Princípios:
  - Modularidade
  - Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo
  - Suporte a tipos abstratos de dados

Conceitos OO

- Uma classe define um novo tipo de dados. Esse tipo pode ser usado para criar objetos desse tipo.
- Uma classe é um modelo para um objeto, e um objeto é uma instância da classe.

Conceitos OO



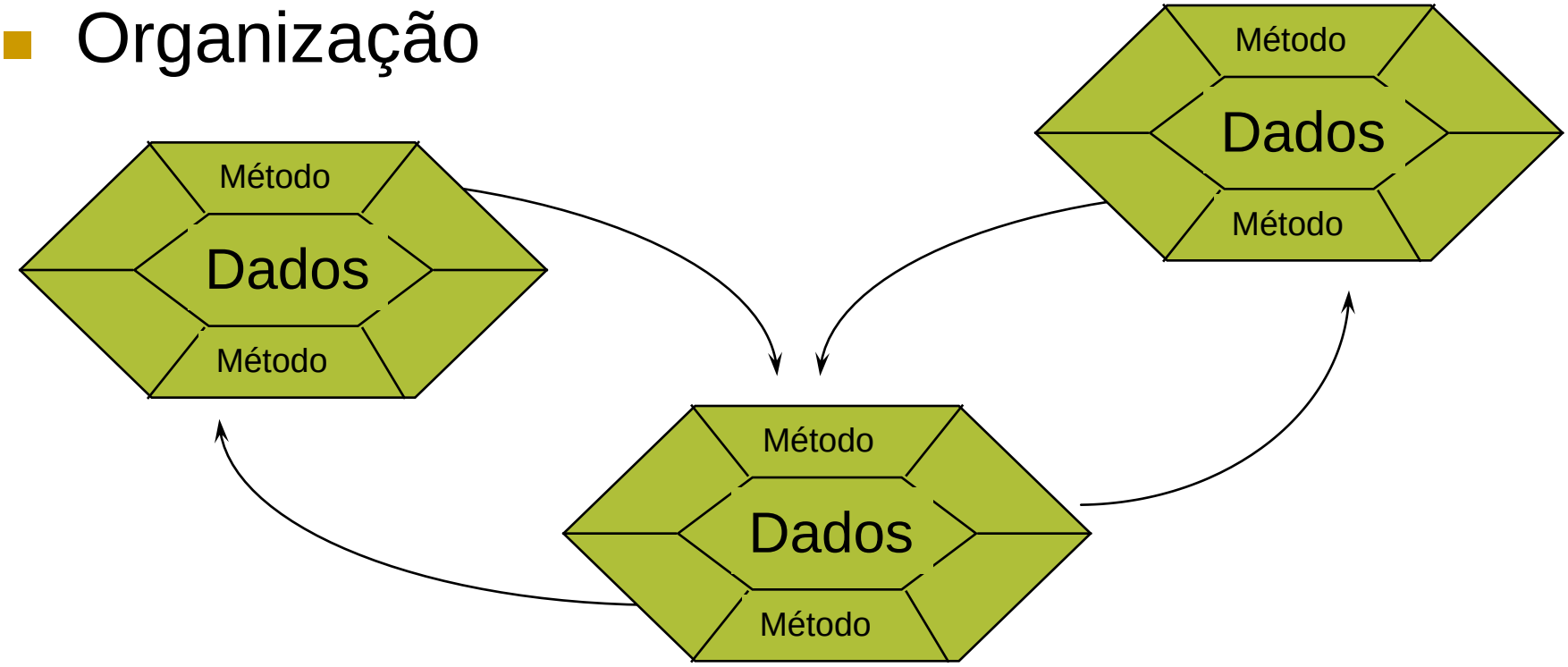
Classes e Objetos

Helicóptero

Inseto

Orientação por objetos

- Organização



Aspectos dinâmico e estático

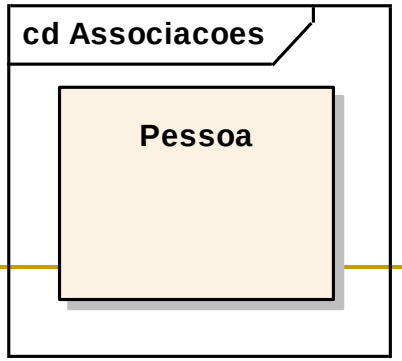
- A funcionalidade externa de um sistema orientado a objetos é fornecida através de colaborações entre objetos.
  - Externamente, os atores visualizam resultados de cálculos, relatórios produzidos, confirmações de requisições realizadas, etc.
  - Internamente, os objetos colaboram uns com os outros para produzir os resultados.
- Essa colaboração pode ser vista sob o ***aspecto dinâmico*** e sob o ***aspecto estrutural estático***.

Modelo de classes

- O diagrama da UML utilizado para representar o aspecto estático é o ***diagrama de classes***.
- O ***modelo de classes*** é composto desse diagrama e da descrição textual associada.
- **Classes de análise** representam um modelo conceitual primário para elementos que têm responsabilidade e comportamento no sistema.
- O objetivo da análise é fazer um mapeamento preliminar do comportamento requerido para os elementos de modelagem no sistema a serem implementados posteriormente.

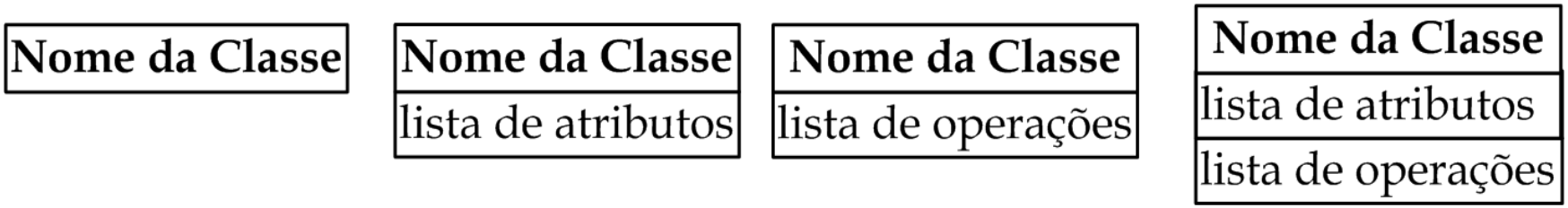
Classes

- Uma classe representa um grupo de objetos semelhantes.
- Uma classe descreve esses objetos através de ***atributos*** e ***operações***.
- Os atributos correspondem às informações que um objeto armazena.
- As operações correspondem às ações que um objeto sabe realizar.

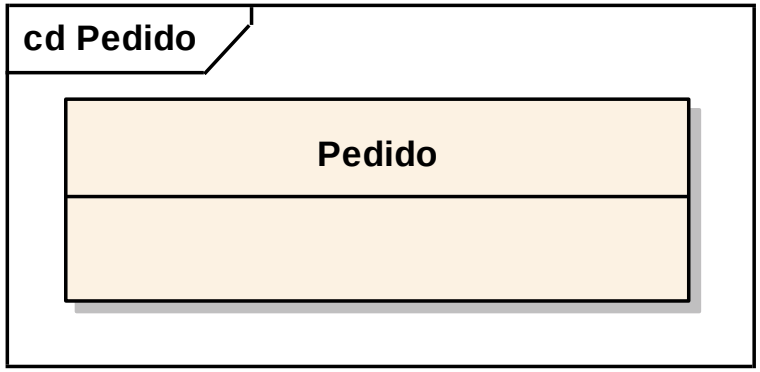


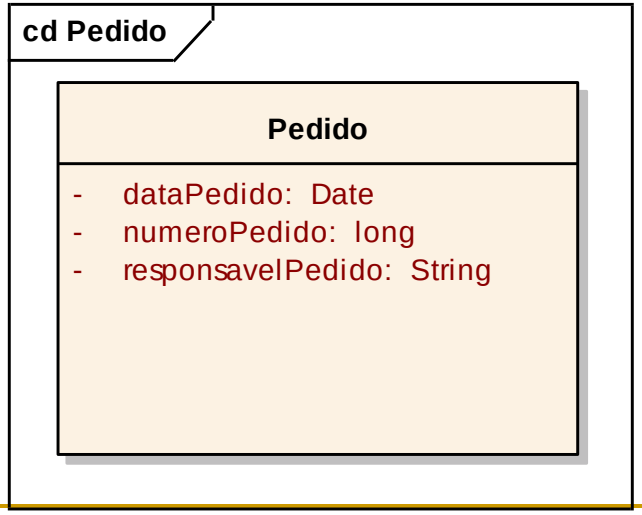
Notação para uma classe

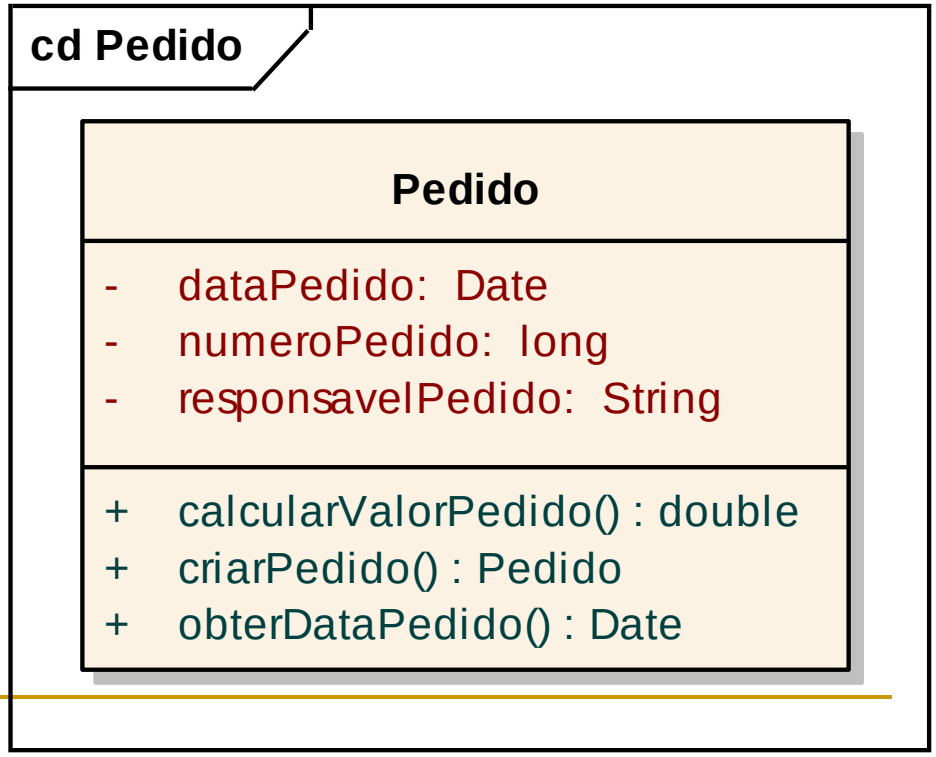
- Representada através de uma “caixa” com no máximo três compartimentos exibidos.
- Notação utilizada depende do nível de abstração desejado.



Exemplo (classe Pedido)





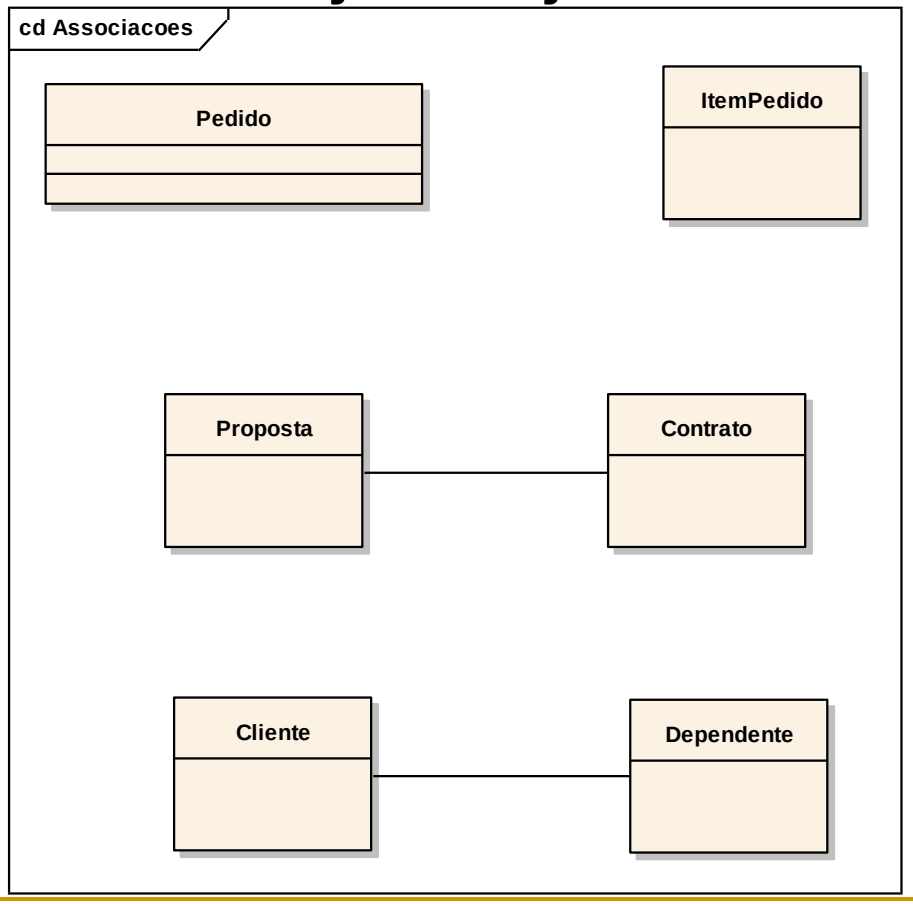


Associações

- Para representar o fato de que objetos podem se relacionar uns com os outros, utiliza-se a *associação*.
- Uma associação representa relacionamentos (ligações) que são formados entre objetos durante a execução do sistema.
  - embora as associações sejam representadas entre classes do diagrama, tais associações representam ligações possíveis entre *objetos* das classes envolvidas.

Notação para uma associação

- Representada através de um segmento de reta ligando as classes cujos objetos se relacionam.
- Exemplos:



Multiplicidades

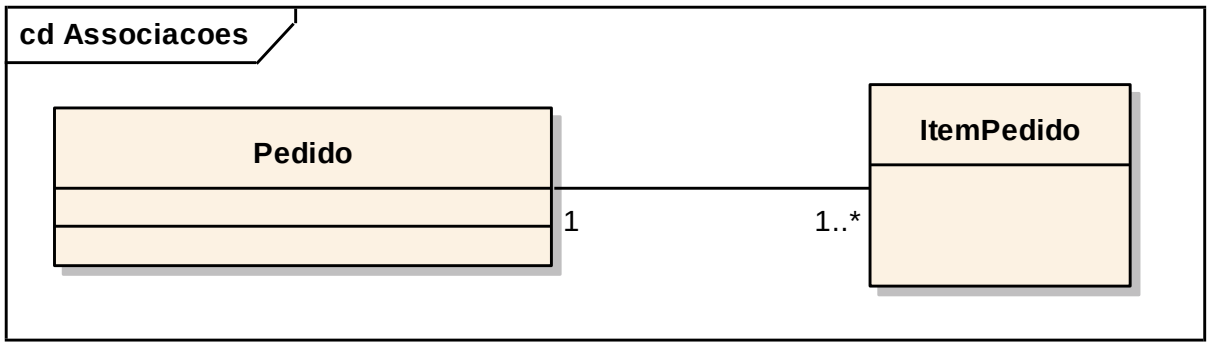
- Representam a informação dos limites inferior e superior da quantidade de objetos aos quais um outro objeto pode estar associado.
- Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha de associação.

Multiplicidades

Nome	**Simbologia**
Apenas Um	1..1 (ou 1)
Zero ou Muitos	0..* (ou *)
Um ou Muitos	1..*
Zero ou Um	0..1

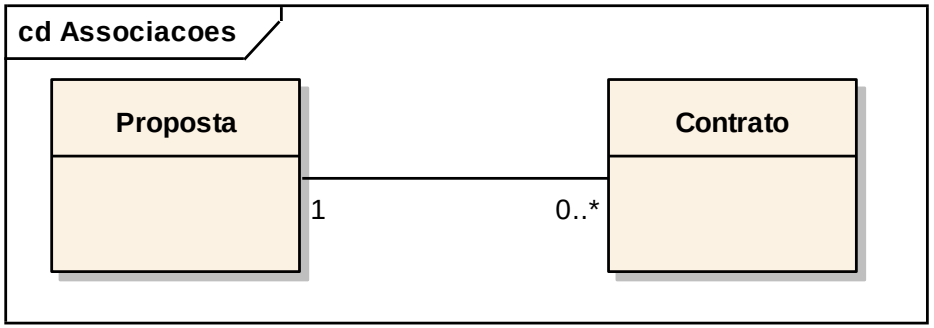
Exemplo (multiplicidade)

- Pode haver um pedido que esteja associado a vários itens.
- Todo pedido está associado a pelo menos um item.
- Um item de pedido está associado a um, e somente um, pedido.



Exemplo (multiplicidade)

- Uma proposta pode não estar associada a nenhum contrato.
- Uma proposta pode estar associada a vários contratos.
- Um contrato está associado a uma e somente uma proposta.



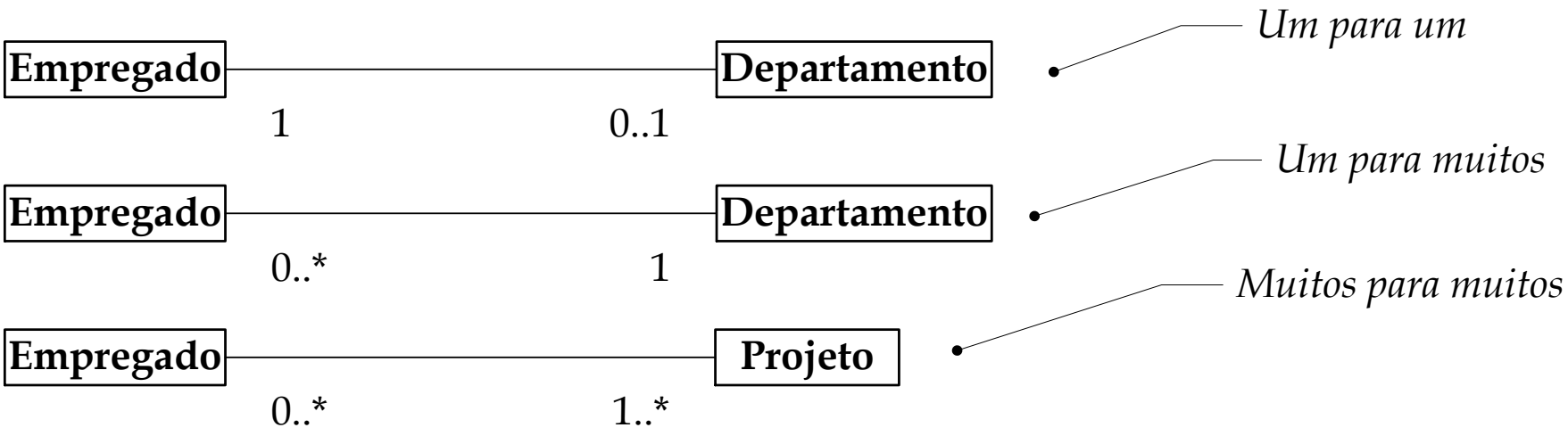
Conectividade

- A **conectividade** corresponde ao tipo de associação entre duas classes: "*muitos para muitos*", "*um para muitos*" e "*um para um*".
- A conectividade da associação entre duas classes depende dos símbolos de multiplicidade que são utilizados na associação.

Conectividade X Multiplicidade

Conectividade	Em um extremo	No outro extremo
Um para um	0..1 1	0..1 1
Um para muitos	0..1 1	* 1..* 0..*
Muitos para muitos	* 1..* 0..*	* 1..* 0..*

Exemplo (conectividade)



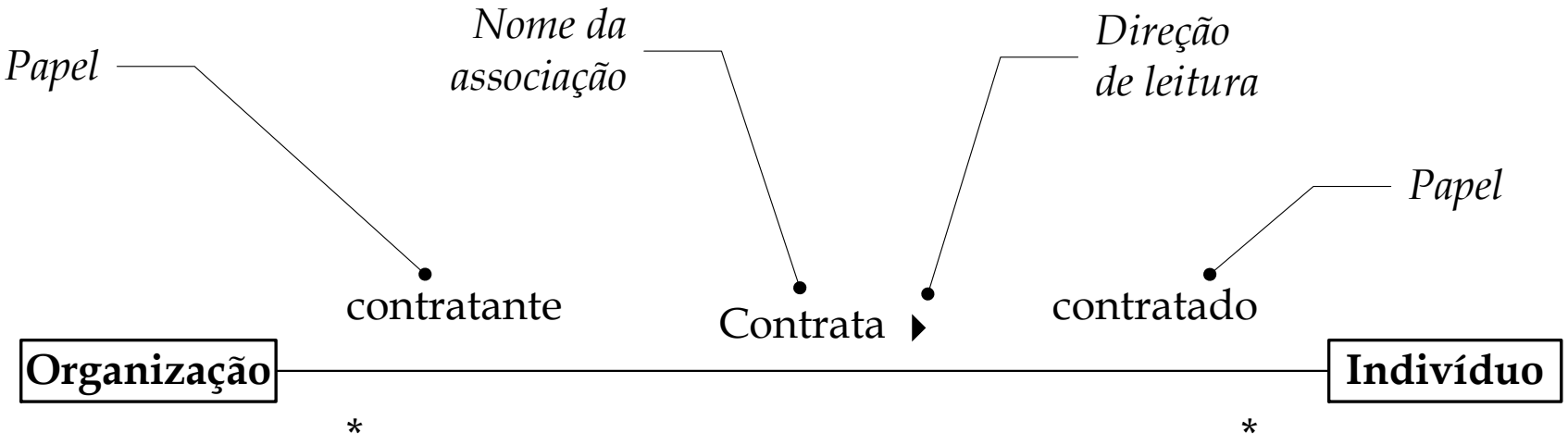
Participação

- Uma característica de uma associação que indica a necessidade (ou não) da existência desta associação entre objetos.
- A participação pode ser *obrigatória* ou *opcional*.
  - Se o valor mínimo da multiplicidade de uma associação é igual a 1 (um), significa que a participação é obrigatória
  - Caso contrário, a participação é opcional.

Nome de associação, direção de leitura e papéis

- Para melhor esclarecer o significado de uma associação no diagrama de classes, a UML define três recursos de notação:
  - ***Nome da associação***: fornece algum significado semântico a mesma.
  - ***Direção de leitura***: indica como a associação deve ser lida
  - ***Papel***: para representar um papel específico em uma associação.

Exemplo (Nome de associação, direção de leitura e papéis)



Agregação

- É um caso especial da associação
  - conseqüentemente, multiplicidades, participações, papéis, etc. podem ser usados igualmente
- Utilizada para representar conexões que guardam uma relação *todo-parte* entre si.
- Em uma agregação, um objeto está contido no outro, ao contrário de uma associação.
- Onde se puder utilizar uma agregação, uma associação também poderá ser utilizada.

Agregação

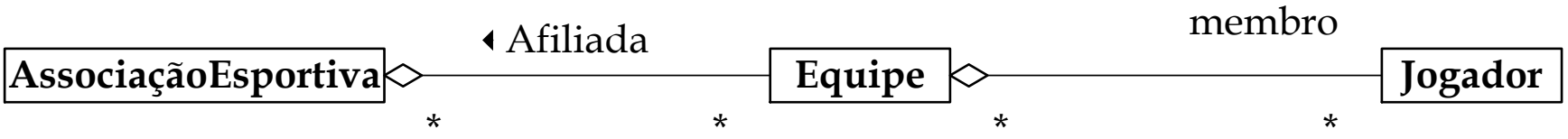
- Características particulares:
  - Agregações são assimétricas: se um objeto A é parte de um objeto B, B não pode ser parte de A.
  - Agregações propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica as suas partes.

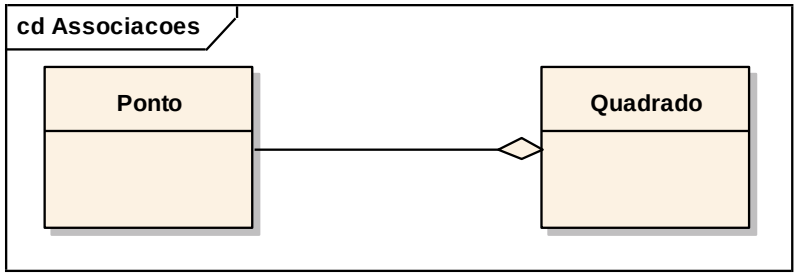
Agregação

- Sejam duas classes associadas, X e Y. Se uma das perguntas a seguir for respondida com um sim, provavelmente há uma agregação onde X é todo e Y é parte.
  - *X tem um ou mais Y?*
  - *Y é parte de X?*

Notação para uma agregação

- *Representada como uma linha conectando as classes relacionadas, com um diamante (losango) branco perto da classe que representa o todo.*
- Exemplo:





Atributos

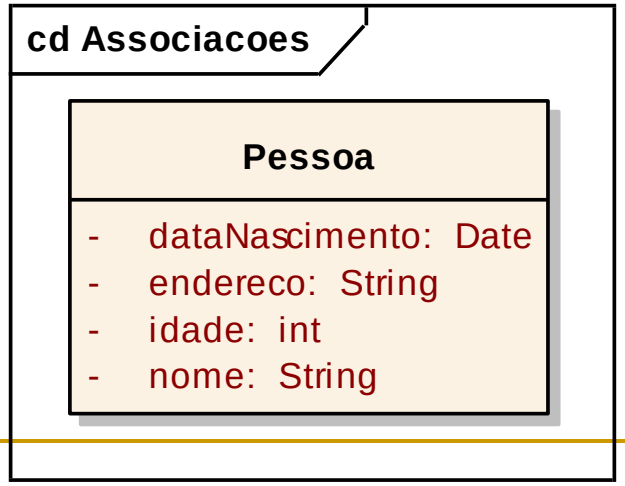
- Atributos são usados para armazenar informações de uma classe
- Exemplos:
  - Nome
  - Endereço
  - Data de processamento
  - Código
  - Quantidade vendida
  - Preço

Atributos

- Depois de definir as primeiras classes do modelo, o próximo passo é identificar os atributos de cada classe.
- O nome do atributo deve ser um substantivo que expressa a informação que o atributo guarda.
- Atributos são de alguns tipos de dados
  - Inteiro, real, booleano
  - Data, Pessoa, Pedido, Empregado

Atributos

- Os casos de uso são a melhor fonte para coletar os atributos de classes.
- Devem estar detalhados o suficiente para isso ou esta informação deve estar armazenada em outro artefato.

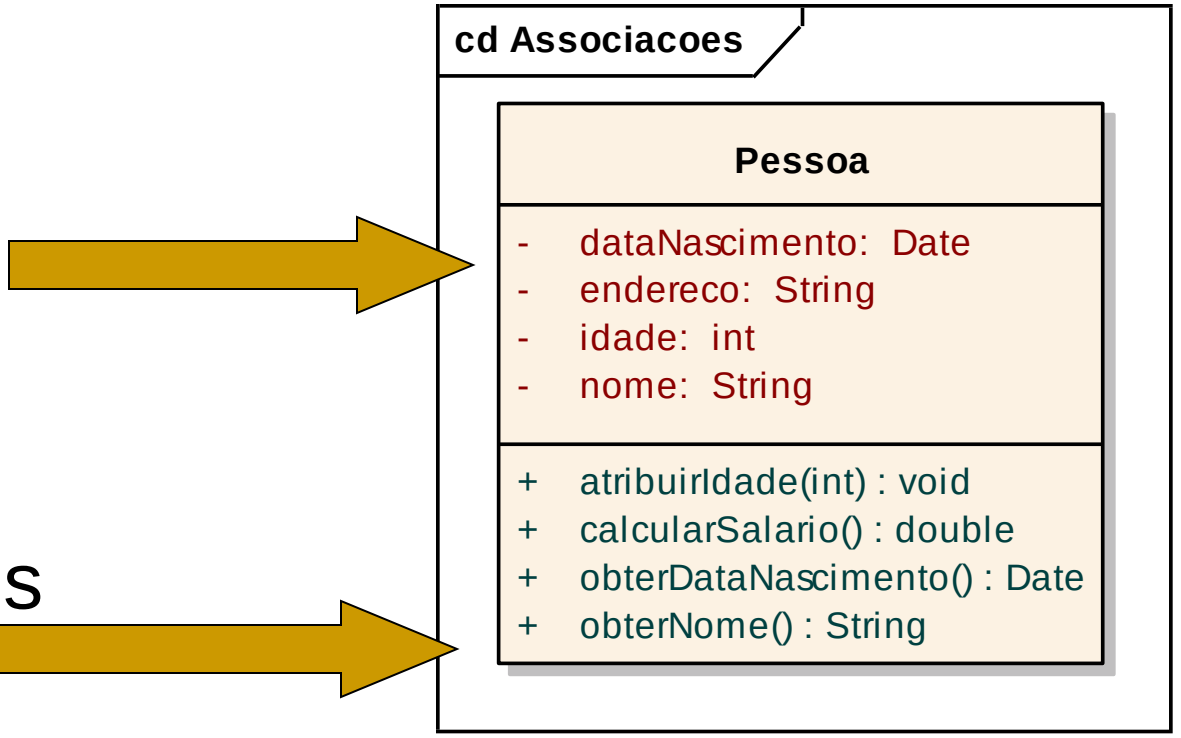


Operações

- Dinâmica do modelo só é possível mediante a troca de mensagens entre os objetos.
- Para um objeto executar um serviço, ele executa uma operação.
- Uma operação pode ter:
  - Nome
  - Tipos de retorno
  - Parâmetros
- É invocada em uma instância da classe da qual ela é um recurso.

Operações - Representação

- Atributos
- Operações

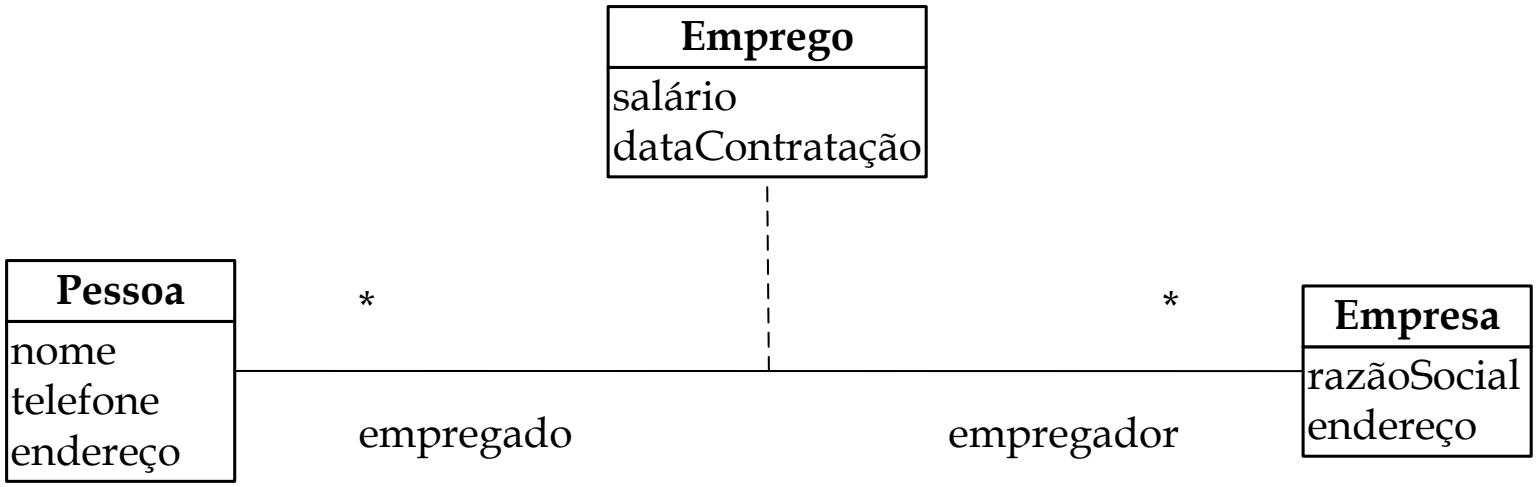


Classe associativa

- É uma classe que está ligada a uma associação, ao invés de estar ligada a outras classes.
- É normalmente necessária quando duas ou mais classes estão associadas, e é necessário manter informações sobre esta associação.
- Uma classe associativa pode estar ligada a associações de qualquer tipo de conectividade.

Notação para uma classe associativa

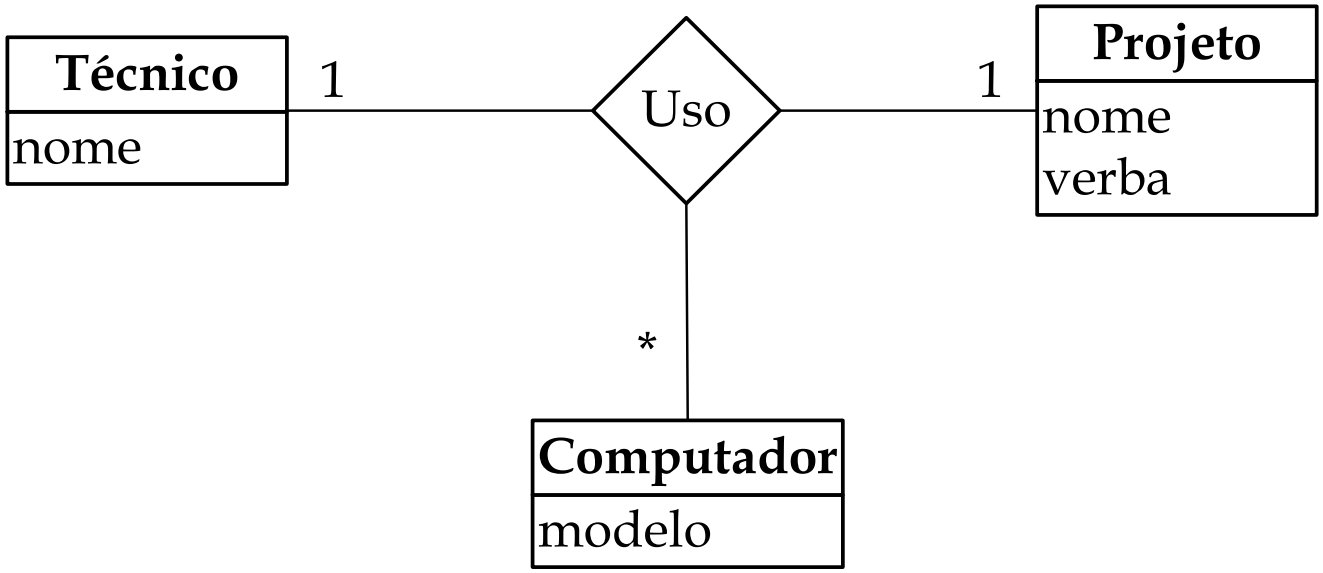
- *Representada pela notação utilizada para uma classe. A diferença é que esta classe é ligada a uma associação.*
- Exemplo:



Associações n-árias

- São utilizadas para representar a associação existente entre objetos de n classes.
- Uma ***associação ternária*** são uma caso mais comum (menos raro) de associação n-ária (n = 3).
- Na notação da UML, as linhas de associação se interceptam em um losango.

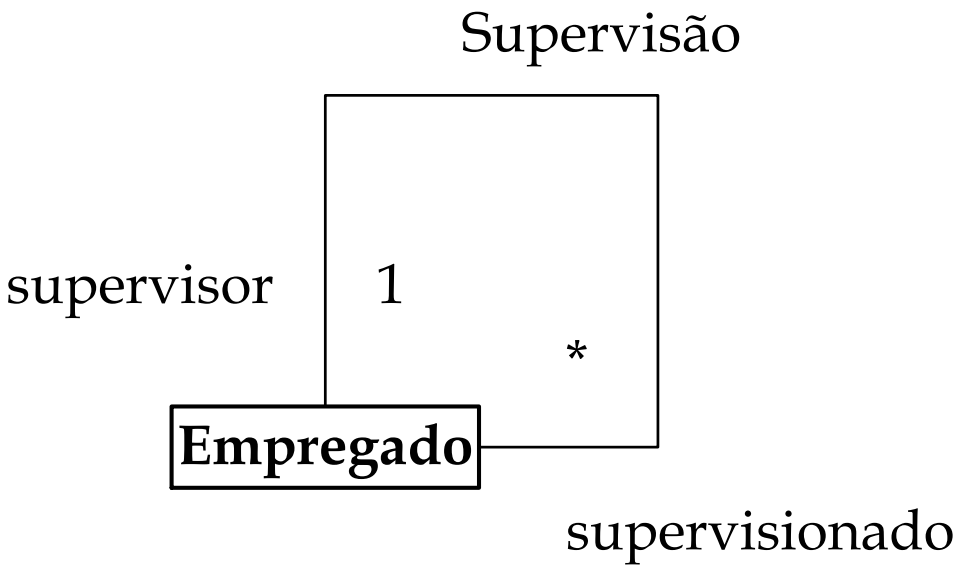
Exemplo (associação ternária)



Associações reflexivas

- Associa objetos da mesma classe.
  - Cada objeto tem um papel distinto na associação.
- A utilização de papéis é bastante importante para evitar ambigüidades na leitura da associação.
- Uma associação reflexiva *não* indica que um objeto se associa com ele próprio.
  - Ao contrário, indica que objetos de uma mesma classe se associam

Exemplo (associação reflexiva)



Herança

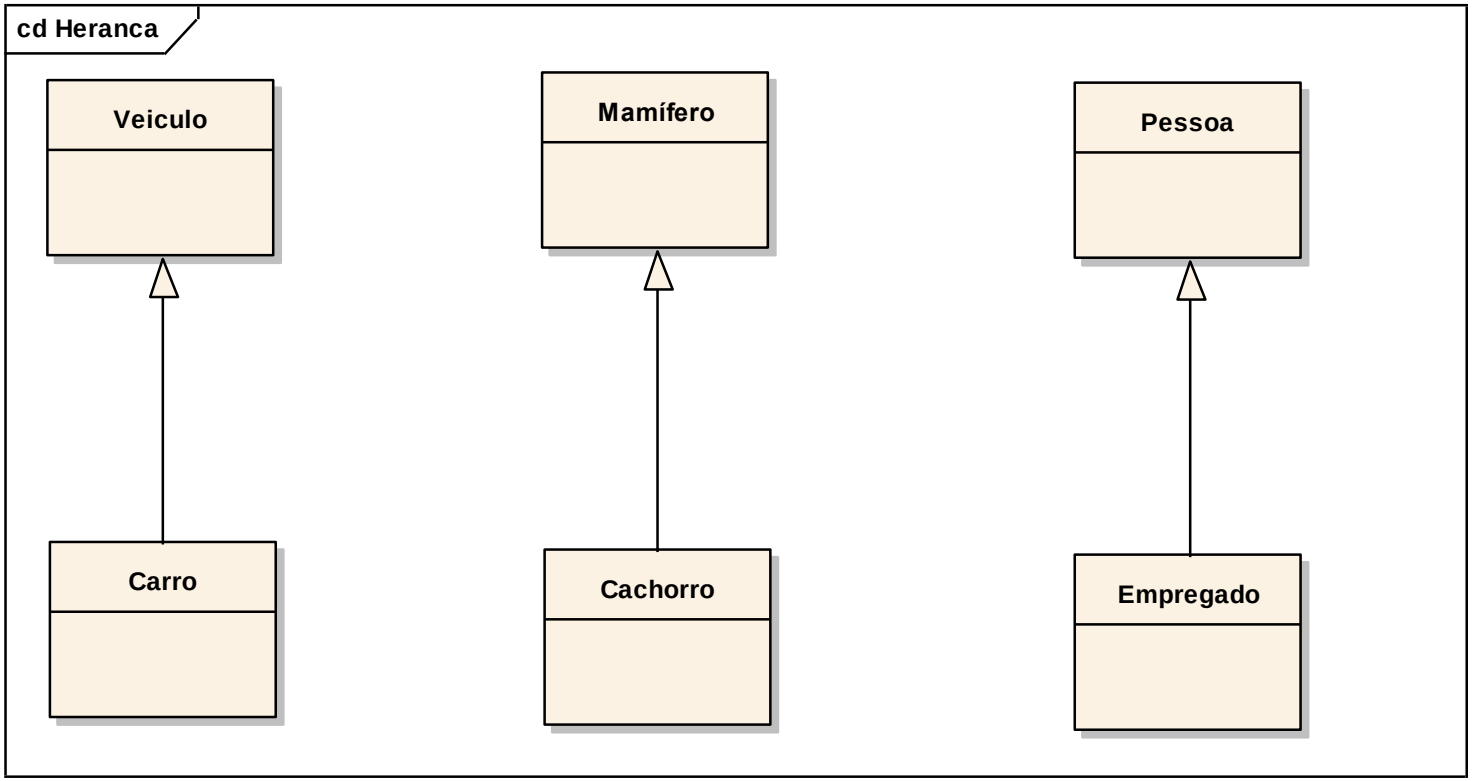
- Um dos alicerces da orientação por objetos
- Também chamada de generalização
- Criação de uma classe mais genérica que define traços comuns a um conjunto de itens
- Relacionamento entre uma classe mais geral e uma mais específica.
- Conceitos:
  - Super classe e sub classe
  - Classe pai e classe filha

Herança

- Permite a criação de novos tipos com propriedades adicionais ao tipo original
- Herança exemplifica um tipo de relacionamento “é um”.
- Exemplos:
  - Mamífero e Cachorro
  - Veículo e Carro
  - Pessoa e Empregado

Herança - Representação

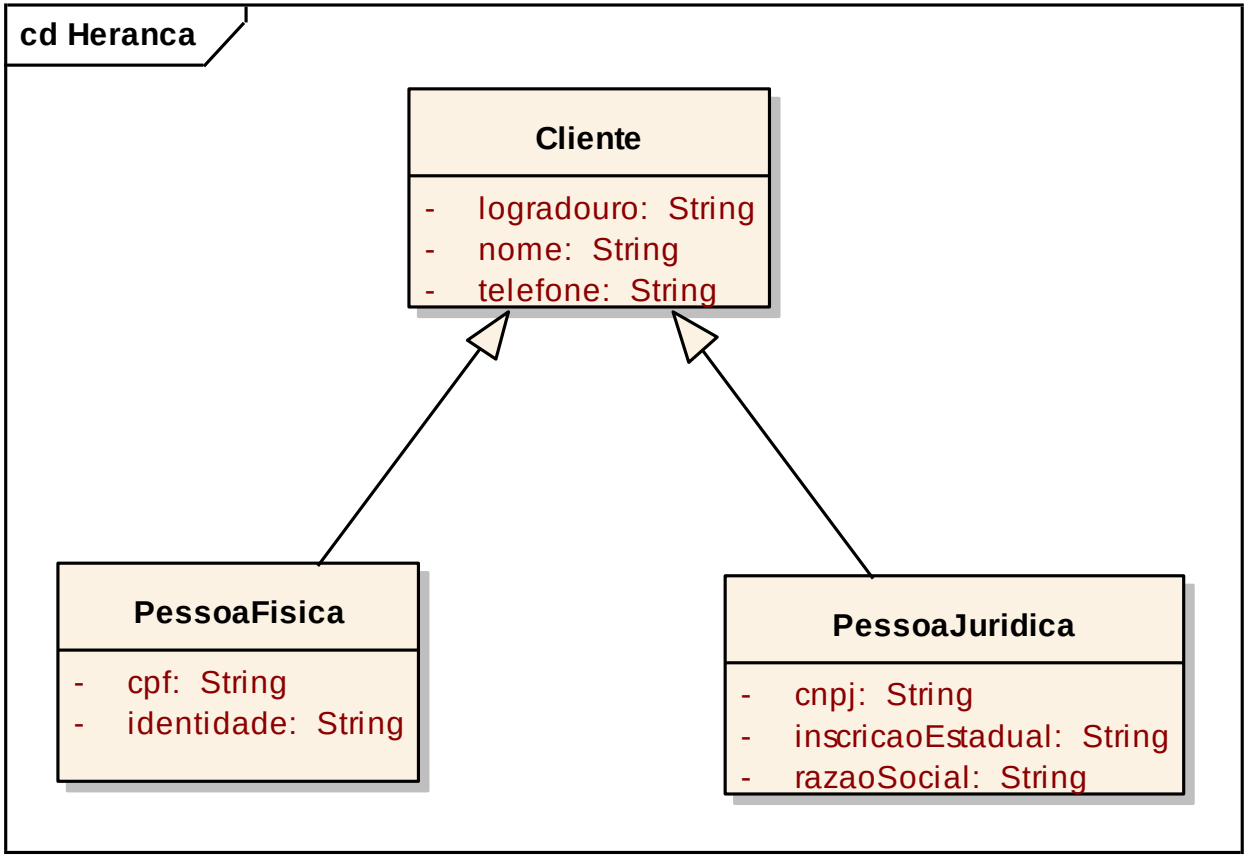
- Um carro “é um” veículo



Herança

- Recursos especificados para instâncias da classe geral são também implicitamente especificados para instâncias da classe específica.
- Herança permite racionalizar o modelo definindo classes específicas:
  - Percebemos que objetos do mundo real apresentam características comuns e outras que lhe são particulares.

Herança - Exemplo



Identificando classes

- Um sistema de software orientado a objetos é composto de objetos em colaboração para realizar as tarefas deste sistema.
- Por outro lado, todo objeto pertence a uma classe.
- Portanto, quando se fala na identificação das classes, o objetivo na verdade é saber quais objetos irão compor o sistema.

Identificando classes

- De uma forma geral, a identificação de classes se divide em duas atividades.
  - Primeiramente, ***classes candidatas*** são identificadas.
  - Depois disso, são aplicados alguns princípios para eliminar classes candidatas desnecessárias.
- Identificar *possíveis* classes para um sistema não é complicado; o difícil é eliminar deste conjunto o que não é necessário.

Identificação dirigida a responsabilidades

- Método de identificação onde a ênfase está na identificação de classes a partir de seus *comportamentos* externos relevantes para o sistema.
  - *como* a classe faz para cumprir com suas responsabilidades deve ser abstraído.
- O esforço recai sobre a identificação das *responsabilidades* que cada classe deve ter.
- "O método *dirigido a responsabilidades enfatiza o encapsulamento da estrutura e do comportamento dos objetos.*".

Responsabilidades e colaboradores

- Em sistemas OO, objetos encapsulam tanto dados quanto comportamento.
- O comportamento de um objeto é definido de tal forma que ele possa cumprir com suas ***responsabilidades***.
- Uma responsabilidade é uma obrigação que um objeto tem para com o sistema no qual ele está inserido.
  - Através delas, um objeto colabora (ajuda) com outros para que os objetivos do sistema sejam alcançados.

Responsabilidades e colaboradores

- Na prática, uma responsabilidade é alguma coisa que um objeto *conhece* ou *faz. (sozinho ou não)*.
  - Um objeto Cliente *conhece* seu nome, seu endereço, seu telefone, etc.
  - Um objeto Pedido *conhece* sua data de realização e sabe *fazer* o cálculo do seu total.
- Se um objeto tem uma responsabilidade com a qual não pode cumprir sozinho, ele deve requisitar ***colaborações*** de outros objetos.

Partindo para a identificação

- ***Análise os casos de uso***: cada caso de uso é analisado para identificar classes candidatas.
- Premissa: a partir da descrição textual dos casos de uso, podem-se derivar as classes do sistema.
  - a existência de uma classe em um sistema só pode se justificar se ela participa de alguma forma para o comportamento externamente visível do sistema.

Partindo para a identificação

- Os substantivos que aparecem no texto do caso de uso são destacados.
  - São também consideradas locuções equivalentes a substantivos.
- Sinônimos são removidos.
- Vantagem: abordagem é bastante simples.
- Desvantagem: depende de como os casos de uso foram escritos.
  - em linguagem natural, as formas de expressar uma mesma idéia são bastante numerosas.

Dicas para atribuição de responsabilidades

- Associar responsabilidades com base na especialidade da classe.
- Distribuir a inteligência do sistema
- Agrupar as responsabilidades conceitualmente relacionadas
- Evitar responsabilidades redundantes

Definição de associações e agregações

- O fato de uma classe possuir colaboradores indica que devem existir relacionamentos entre estes últimos e a classe.
  - Isto porque um objeto precisa conhecer o outro para poder lhe fazer requisições.
  - Portanto, para criar associações, verifique os colaboradores de uma classe.
- O raciocínio para definir associações reflexivas, ternárias e agregações é o mesmo.

Documentando o modelo de classes

- As responsabilidades e colaboradores mapeados para elementos do modelo de classes devem ser organizados em um diagrama de classes e documentados, resultando no ***modelo de classes de domínio***.
- Podem ser associados estereótipos predefinidos às classes identificadas:
  - <<fronteira>>
  - <<entidade>>
  - <<controle>>

Documentando o modelo de classes

- O modelador pode optar por “esconder” as classes de fronteira ou até mesmo as classes de controle.
  - Uma ferramenta CASE que dê suporte a essa operação seria de grande ajuda para a equipe de desenvolvimento.
- Além do diagrama, as classes devem ser documentadas textualmente.

Modelo de classes no processo de desenvolvimento

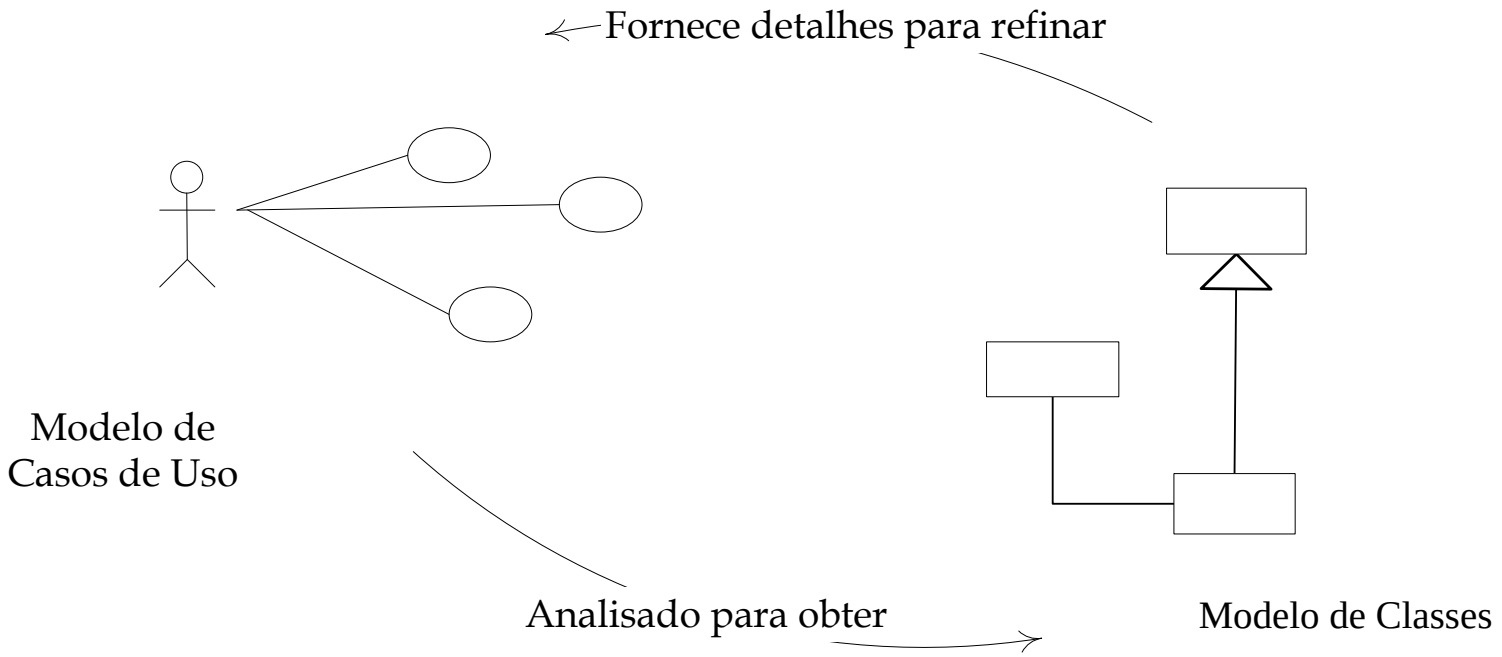
- Em um desenvolvimento dirigido a casos de uso, após a descrição dos casos de uso, é possível iniciar a identificação de classes.
- As classes identificadas são refinadas para retirar inconsistências e redundâncias.
- As classes são documentadas e o diagrama de classes inicial é construído, resultando no modelo de classes de domínio.

Modelo de classes no processo de desenvolvimento

- Inconsistências nos modelos devem ser verificadas e corrigidas.
- As construções do modelo de casos de uso e do modelo de classes são retroativas uma sobre a outra.
  - Pode-se identificar a necessidade de modificação de casos de uso preexistentes.

Modelo de classes no processo de desenvolvimento

- Detalhes são adicionados aos modelos, à medida que o problema é entendido.



Objetos de Entidade

- Um objeto de entidade é um repositório para alguma ***informação*** manipulada pelo sistema.
- Esses objetos representam conceitos do domínio do negócio.
- Normalmente esses objetos armazenam informações persistentes.
- Há várias instâncias de uma mesma classe de entidade coexistindo no sistema.

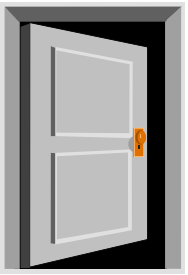
Objetos de Entidade

- Atores não têm acesso direto a estes objetos.
  - Objetos de entidade se comunicam com o exterior do sistema por intermédio de outros objetos.
- Objetos de entidade normalmente participam de vários casos de uso e têm um ciclo de vida longo.
  - Um objeto *Pedido* pode participar dos casos de uso *Realizar Pedido* e *Atualizar Estoque*. Este objeto pode existir por diversos anos ou mesmo tanto quanto o próprio sistema.

Objetos de Entidade

- Responsabilidades de fazer típicas de objetos de entidade:
  - Informar valores de seus atributos a objetos de controle.
  - Realizar cálculos simples, normalmente com a colaboração de objetos de entidade associados através de agregações.
  - Criar e destruir objetos parte (considerando que o objeto de entidade seja um objeto todo de uma agregação).

Objetos de Fronteira



- Esses objetos traduzem os eventos gerados por um ator em eventos relevantes ao sistema.
- Também são responsáveis por apresentar os resultados de uma interação dos objetos em algo inteligível pelo ator.
- Um objeto de fronteira existe para que o sistema se comunique com o mundo exterior.
- Por conseqüência, estes objetos são altamente dependentes do ambiente.

Objetos de Fronteira

- Classes de fronteira realizam a comunicação do sistema com atores, sejam eles outros sistemas, equipamentos ou seres humanos.
- Há três tipos principais de classes de fronteira:
  - as que se comunicam com o usuário (atores humanos),
  - as que se comunicam com outros sistemas
  - as que se comunicam com dispositivos atrelados ao sistema.

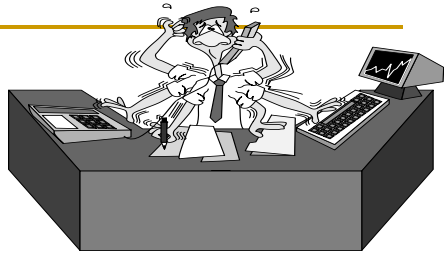
Objetos de Fronteira

- Tipicamente têm as seguintes responsabilidades *de fazer*:
  - Notificar aos objetos de controle de eventos gerados externamente ao sistema.
  - Notificar aos atores do resultado de interações entre os objetos internos.

Objetos de Fronteira

- Responsabilidades *de conhecer* de classes de fronteira para interação humana representam informação manipulada através da interface com o usuário.
  - A construção de protótipos pode ajudar a identificar essas responsabilidades.
- Responsabilidades de conhecer para classes de fronteira que realizam comunicação com outros sistemas representam propriedades de uma interface de comunicação.

Objetos de Controle



- São a “ponte de comunicação” entre objetos de fronteira e objetos de entidade.
- Responsáveis por controlar a lógica de execução correspondente a um caso de uso.
- Decidem o que o sistema deve fazer quando um evento externo relevante ocorre.
  - Eles realizam o controle do processamento
  - Agem como ***gerentes*** (coordenadores, controladores) dos outros objetos para a realização de um ou mais caso de uso.

Objetos de Controle

- São bastante acoplados à lógica da aplicação.
- Traduzem eventos externos em operações que devem ser realizadas pelos demais objetos.
- Ao contrário dos objetos de entidade e de fronteira, objetos de controle são tipicamente ativos
  - consultam informações e requisitam serviços de outros objetos.

Objetos de Controle

- Responsabilidades de fazer típicas:
  - Realizar monitorações, a fim de responder a eventos externos ao sistema (gerados por objetos de fronteira).
  - Coordenar a realização de um caso de uso através do envio de mensagens a objetos de fronteira e objetos de entidade.
  - Assegurar que as regras do negócio (Seção 4.5.1) estão sendo seguidas corretamente.
  - Coordenar a criação de associações entre objetos de entidade.

Objetos de Controle

- Responsabilidades de conhecer estão associadas manter valores acumulados, temporários ou derivados durante a realização de um caso de uso.
- Podem também ter o objetivo de manter o estado da realização do caso de uso.
- Têm vida curta: normalmente existem somente durante a realização de um caso de uso.

Divisão de responsabilidades

- A categorização de responsabilidades implica em que cada objeto é especialista em realizar um de três tipos de tarefa:
  - se comunicar com atores (fronteira)
  - manter as informações do sistema (entidade)
  - coordenar a realização de um caso de uso (controle).
- A importância dessa categorização está relacionada à capacidade de adaptação do sistema a eventuais mudanças

Divisão de responsabilidades

- Se cada objeto tem funções específicas dentro do sistema, eventuais mudanças no sistema podem ser:
  1. menos complexas
  2. mais localizadas.
- Uma eventual modificação em uma parte do sistema tem menos possibilidades de resultar em mudanças em outras partes.

Divisão de responsabilidades

Tipo de mudança	Onde mudar
Mudanças em relação à interface gráfica, ou em relação à comunicação com outros sistemas.	Fronteira
Mudanças nas informações manipuladas pelo do sistema	Entidade
Mudanças em funcionalidades complexas (lógica do negócio)	Controle

Divisão de responsabilidades

- Exemplo: vantagem de separação de responsabilidades em um sistema para uma loja de aluguel de carros.
  - Se o sistema tiver que ser atualizado para que seus usuários possam utilizá-lo pela Internet, a lógica da aplicação não precisaria de modificações.
  - Considerando a lógica para calcular o valor total das locações feitas por um cliente: se esta lógica estiver encapsulada em uma classe de controle, somente esta classe precisaria de modificação.

Divisão de responsabilidades

- A construção de um sistema de software que faça separação das responsabilidades de apresentação (fronteira), de lógica da aplicação (controle) e de manutenção dos dados (entidade):
  - facilita também o reuso dos objetos no desenvolvimento de sistemas de software semelhantes.
  - ajuda no desacoplamento entre elementos do sistema