**DSS – Diagrama de Sequência de Sistema**

Também conhecido como cenário.

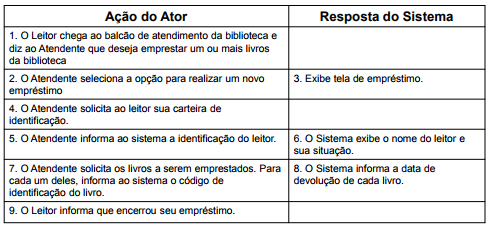
* Mostra um cenário global do funcionamento do sistema.

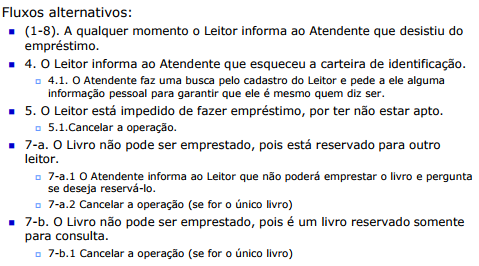
Mostra, em alto nível, os principais eventos (de entrada ou de saída) que fazem parte de um caso de uso.

* Cada um desses eventos dispara uma operação do sistema para tratá-lo.
* A ideia é identificar quais são esses eventos e quem é o ator responsável por iniciá-lo.

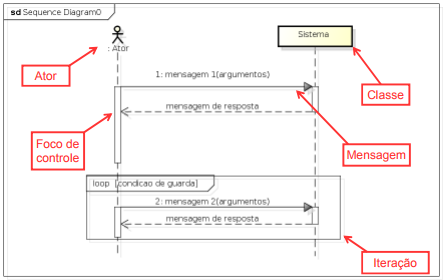
Caso de uso “Emprestar Livro” (de uma biblioteca).

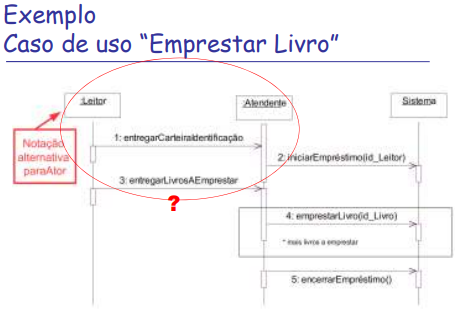
Fluxo principal:



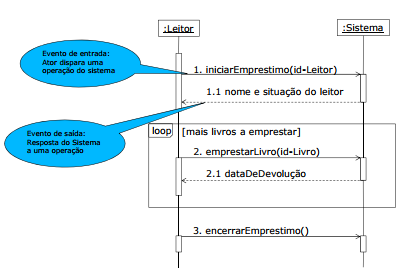


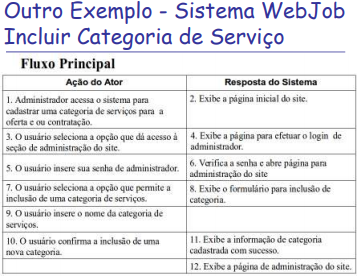
* Considerando esse caso de uso, ocorrem basicamente 3 eventos:
  + Iniciar o empréstimo, quando o Atendente informa ao sistema a identificação do Leitor para checar se é um Leitor apto a emprestar;
  + Emprestar livro(s), em que são informados ao sistema os dados sobre os livros a emprestar;
  + e Encerrar o empréstimo, em que registra-se efetivamente o empréstimo para o Leitor, mudando a situação dos livros para “em circulação”.
* Diz-se que “iniciar o empréstimo” é um evento de entrada que ocorre e dispara a execução de uma operação para tratá-lo.
* Elementos:
  + Atores
  + Classes
  + Mensagens
  + Linhas de vida e focos de controle
  + Iterações

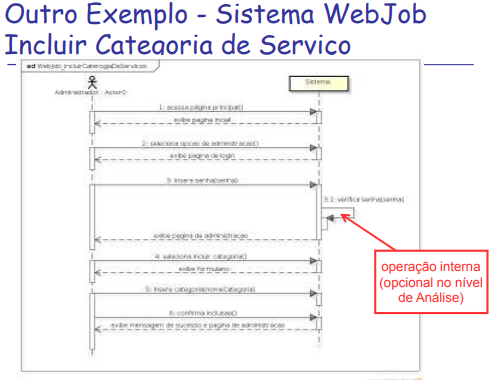




* Eventos envolvendo dois atores (como o evento entregarCarteiraIdentificação do slide anterior) são desconsiderados pois: Ficam fora dos limites do sistema. Não se tornarão operações efetivamente implementadas no software.







Na fase de projeto, o interesse recai sobre refinar os modelos de análise. Objetivo: encontrar alternativas para que o software atenda aos requisitos funcionais, ao mesmo tempo em que respeite as restrições definidas pelos requisitos não-funcionais.

**Modelo Conceitual**

As funcionalidades de um software OO são realizadas internamente 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.

O diagrama de classes representa o aspecto estrutural e estático que compõe a solução encapsulada em um software OO.

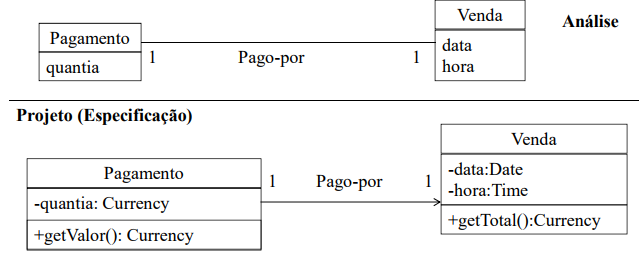
O diagrama de objetos pode ser visto com uma instância de diagramas de classes. Provê o aspecto dinâmico, representado por uma “fotografia” do sistema em certo momento, que pode se modificar a cada instante diferente da execução do software.

Esses diagramas compõem o modelo de objetos do software e evoluem durante o desenvolvimento do software.

Há três níveis sucessivos de detalhamento: Análise > Especificação (Projeto) > Implementação

Modelo de Classes de Análise:

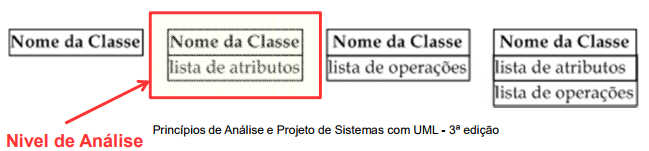
* Representa termos do domínio do negócio. Ideias, coisas, e conceitos no mundo real.
* Objetivo: descrever o problema representado pelo sistema a ser desenvolvido, sem considerar características da solução a ser utilizada.
* É um dicionário “visual” de conceitos e informações relevantes ao sistema sendo desenvolvido.
* Elementos de notação do diagrama de classes normalmente usados na construção do modelo de análise: classes e atributos; associações, composições e agregações (com seus adornos); classes de associação; generalizações (herança).
* O modelo de análise não representa detalhes da solução do problema. Embora este sirva de ponto de partida para uma posterior definição das classes de software (especificação).

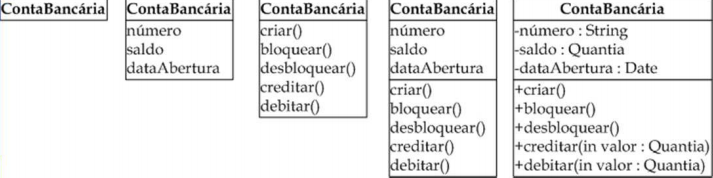


**Diagrama de Classes**

Classes:

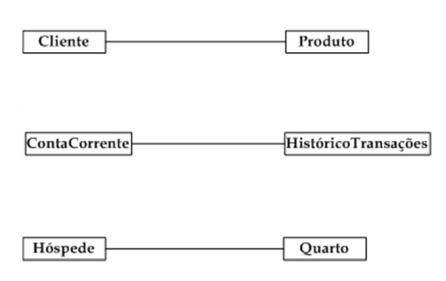
* Uma classe descreve esses objetos através de atributos e operações. Atributos correspondem às informações que um objeto armazena. Operações correspondem às ações que um objeto sabe realizar.
* Notação na UML: “caixa” com no máximo três compartimentos exibidos. Detalhamento utilizado depende do estágio de desenvolvimento e do nível de abstração desejado.





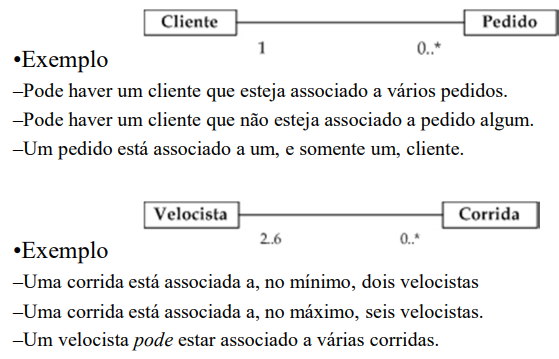
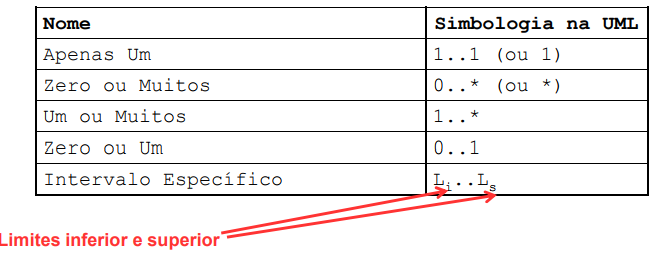
Associações:

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



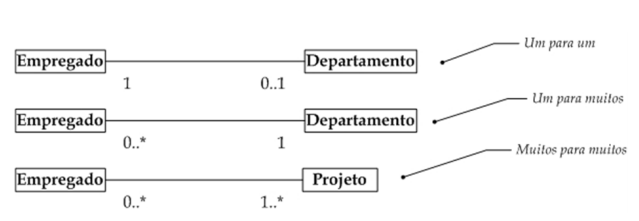
Multiplicidades:

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



Conectividade:

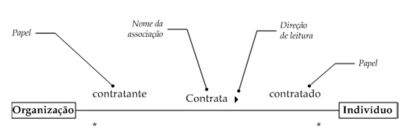
* A conectividade corresponde ao tipo de associação entre duas classes: “muitos para muitos”, “um para muitos” e “um para um”.
* A conectividade da associação entre duas classes depende dos símbolos de multiplicidade que são utilizados na associação.



Participação:

* É 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.

Acessórios para Associações:



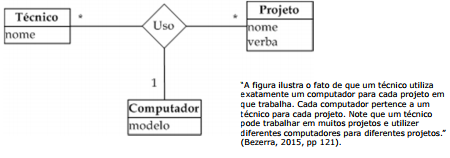
Classe associativa:

* É uma classe que está ligada a uma associação, em vez 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.
* Sinônimo: classe de associação
* A notação é semelhante à utilizada para classes ordinárias. A diferença é que esta classe é ligada a uma associação por uma linha tracejada.
* Exemplo: para cada par de objetos [pessoa, empresa], há duas informações associadas: salário e data de contratação



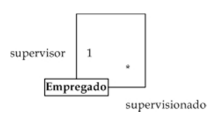
Associações n-árias:

* Define-se o grau de uma associação como a quantidade de classes envolvidas na mesma.
* Na notação da UML, as linhas de uma associação n- ária se interceptam em um losango.
* Na grande maioria dos casos práticos de modelagem, as associações normalmente são binárias.
* Quando o grau de uma associação é igual a três, dizemos que a mesma é ternária. Uma associação ternária é o caso mais comum (menos raro) de associação n-ária (n = 3).
* Na notação da UML, as linhas de uma associação n- ária se interceptam em um losango nomeado.



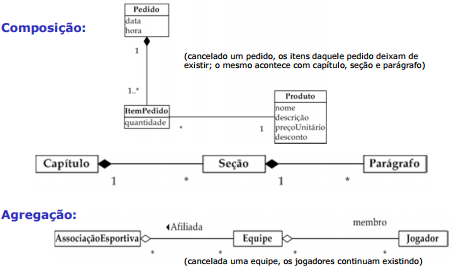
Associação Reflexiva:

* É um tipo especial de associação que representa ligações entre objetos que pertencem a uma mesma classe.
* Não indica que um objeto se associa a ele próprio.
* Quando se usa associações reflexivas, a definição de papéis é importante para evitar ambiguidades na leitura da associação.
* Cada objeto tem um papel distinto na associação.



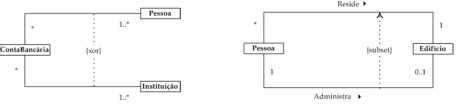
Agregações e Composições:

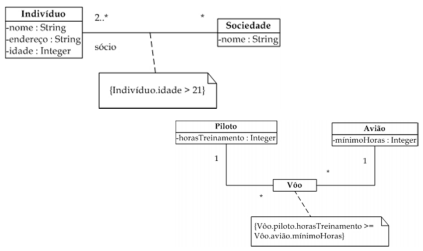
* A semântica de uma associação corresponde ao seu significado, ou seja, à natureza conceitual da relação que existe entre os objetos que participam daquela associação.
* De todos os significados diferentes que uma associação pode ter, há uma categoria especial de significados, que representa relações todo-parte.
* Uma relação todo-parte entre dois objetos indica que um dos objetos está contido no outro. Podemos também dizer que um objeto contém o outro.
* A UML define dois tipos de relacionamentos todo-parte, a agregação e a composição.
* Algumas particularidades das agregações/ composições:
  + são assimétricas, no sentido de que, se um objeto A é parte de um objeto B, o objeto B não pode ser parte do objeto A.
  + propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica às suas partes.
  + as partes são normalmente criadas e destruídas pelo todo. Na classe do objeto todo, são definidas operações para adicionar e remover as partes.
* As diferenças mais marcantes entre elas são:
  + Destruição de objetos: Na agregação, a destruição de um objeto todo não implica necessariamente na destruição do objeto parte.
  + Pertinência: Na composição, os objetos parte pertencem a um único todo. Por essa razão, a composição é também denominada agregação não-compartilhada. Em uma agregação, pode ser que um mesmo objeto participe como componente de vários outros objetos. Por essa razão, a agregação é também denominada agregação compartilhada.



Restrições sobre associações:

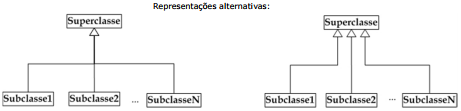
* Restrições OCL (Object Constraint Language) podem ser adicionadas sobre uma associação para adicionar a ela mais semântica.
* Duas das restrições sobre associações predefinidas pela UML são subset e xor.
* O modelador também pode definir suas próprias restrições em OCL.



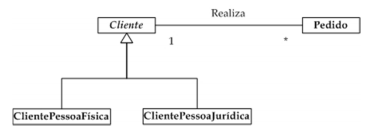


Generalizações e Especializações:

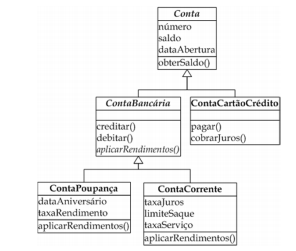
* O modelador também pode representar relacionamentos entre classes. Esses denotam relações de generalidade ou especificidade entre as classes envolvidas.
* Exemplos:
  + o conceito mamífero é mais genérico que o conceito ser humano.
  + o conceito carro é mais específico que o conceito veículo.
* Esse é o chamado relacionamento de herança. relacionamento de generalização/especialização
* Terminologia:
  + subclasse X superclasse
  + supertipo X subtipo
  + classe base X classe herdeira
  + classe de especialização X classe de generalização
  + ancestral e descendente (herança em vários níveis)



* Subclasses herdam as características de sua superclasse. É como se as características da superclasse estivessem definidas também nas suas subclasses. Além disso, essa herança é transitiva e assimétrica.
* Não somente atributos e operações, mas também associações são herdadas pelas subclasses.
* No exemplo abaixo, cada subclasse está associada a Pedido, por herança.

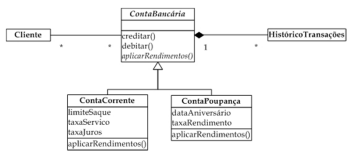


* Transitividade: uma classe em uma hierarquia herda propriedades e relacionamentos de todos os seus ancestrais. Ou seja, a herança pode ser aplicada em vários níveis, dando origem a hierarquia de generalização. Uma classe que herda propriedades de outra classe pode ela própria servir como superclasse.
* Assimetria: dadas duas classes A e B, se A for uma generalização de B, então B não pode ser uma generalização de A. Ou seja, não pode haver ciclos em uma hierarquia de generalização.



Classes abstratas:

* Usualmente, a existência de uma classe se justifica pelo fato de haver a possibilidade de gerar instâncias da mesma. Essas são as classes concretas.
* No entanto, podem existir classes que não geram instâncias diretas. Essas são as classes abstratas.
* Classes abstratas são utilizadas para organizar e simplificar uma hierarquia de generalização. Propriedades comuns a diversas classes podem ser organizadas e definidas em uma classe abstrata, a partir da qual as primeiras herdam.
* Subclasses de uma classe abstrata também podem ser abstratas, mas a hierarquia deve terminar em uma ou mais classes concretas.
* Na UML, uma classe abstrata é representada com o seu nome em itálico. No exemplo a seguir, ContaBancária é uma classe abstrata.

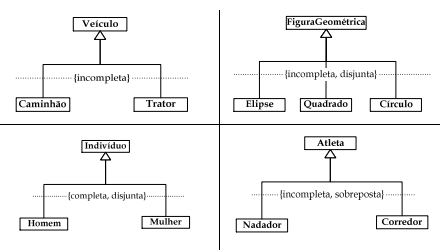


Refinamento do Modelo com Herança:

* Critérios a avaliar na criação de subclasses:
  + A subclasse tem atributos adicionais.
  + A subclasse tem associações.
  + A subclasse é manipulada (ou reage) de forma diferente da superclasse.
* Se algum “subconceito” (subconjunto de objetos) atenda a dos critérios acima, a criação de uma subclasse deve ser considerada.
  + Sempre se assegure de que se trata de um relacionamento do tipo “é-um”: “X é um tipo de Y?” (se sim, é provável que X deva ser definida como uma subclasse de Y)
* A regra “é-um” é mais formalmente conhecida como regra da substituição ou princípio de Liskov.
* Regra da Substituição: sejam duas classes A e B, onde A é uma generalização de B. Não pode haver diferenças entre utilizar instâncias de B ou de A, do ponto de vista dos clientes de A.

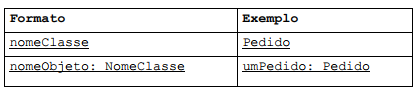
Restrições Sobre Generalização/Especialização:

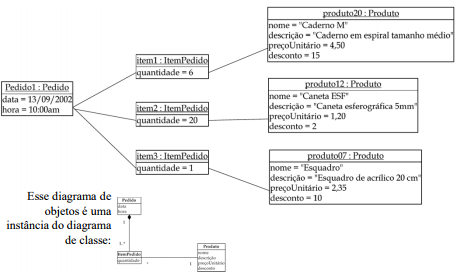
* Restrições OCL sobre relacionamentos de herança podem ser representadas no diagrama de classes, também com o objetivo de esclarecer seu significado.
* Restrições predefinidas pela UML:
  + Sobreposta X Disjunta: Sobreposta permite herança múltipla dentro da hierarquia. Disjunta: não permite herança múltipla dentro da hierarquia
  + Completa X Incompleta: Na completa todas as subclasses já foram previstas. Na incompleta outras subclasses podem surgir.

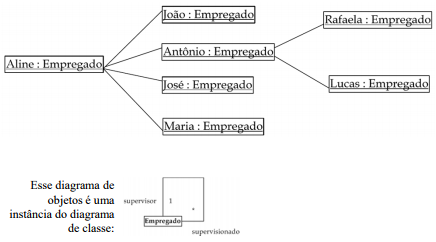


**Diagrama de Objetos**

* Além do diagrama de classes, a UML define um segundo tipo de diagrama estrutural, o diagrama de objetos.
* Pode ser visto com uma instância de diagramas de classes.
* Representa uma “fotografia” do sistema em um certo momento.
* Exibe as ligações formadas entre objetos conforme estes interagem e os valores dos seus atributos.







**Técnicas para Identificação de Classes**

Várias técnicas (de uso não exclusivo) são usadas para identificar classes:

* Categorias de Conceitos
* Análise Textual de Abbott (Abbot Textual Analysis)
* Análise de Casos de Uso
  + (Categorização BCE)
* Padrões de Análise (Analisys Patterns)
* Identificação Dirigida a Responsabilidades

Categorias de Conceitos:

* Estratégia: usar uma lista de conceitos comuns.
* Conceitos concretos. Por exemplo, edifícios, carros, salas de aula, etc.
* Papéis desempenhados por seres humanos. Por exemplo, professores, alunos, empregados, clientes, etc.
* Eventos, ou seja, ocorrências em uma data e em uma hora particulares. Por exemplo, reuniões, pedidos, aterrisagens, aulas, etc.
* Lugares: áreas reservadas para pessoas ou coisas. Por exemplo: escritórios, filiais, locais de pouso, salas de aula, etc.
* Organizações: coleções de pessoas ou de recursos. Por exemplo: departamentos, projetos, campanhas, turmas, etc.
* Conceitos abstratos: princípios ou idéias não tangíveis. Por exemplo: reservas, vendas, inscrições, etc.

Análise Textual de Abbott:

* Estratégia: identificar termos da narrativa de casos de uso e documento de requisitos que podem sugerir classes, atributos, operações.
* São utilizados diversos artefatos sobre o sistema: documento e requisitos, modelos do negócio, glossários, conhecimento sobre o domínio, etc.
* Para cada um desses artefatos, os nomes (substantivos e locuções equivalentes a substantivos) que aparecem no mesmo são destacados.
* Após isso, os sinônimos são removidos (permanecem os nomes mais significativos para o domínio do negócio em questão).
* Cada termo remanescente se encaixa em uma das situações a seguir:
  + O termo se torna uma classe (ou seja, são classes candidatas)
  + O termo se torna um atributo
  + O termo não tem relevância alguma com software pretendido
* Abbott também preconiza o uso de sua técnica na identificação de operações e de associações. Para isso, ele sugere que sejam destacados os verbos no texto.
  + Verbos de ação (e.g., calcular, confirmar, cancelar, comprar, fechar, estimar, depositar, sacar, etc.) são operações em potencial.
  + Verbos com sentido de “ter” são potenciais agregações ou composições.
  + Verbos com sentido de “ser” são generalizações em potencial.
  + Demais verbos são associações em potencial.
* Apesar da simplicidade, uma desvantagem da técnica é que seu resultado (as classes candidatas identificadas) depende da completude dos documentos utilizados como fonte. Dependendo do estilo que foi utilizado para escrever esse documento, essa técnica pode levar à identificação de diversas classes candidatas que não gerarão classes. A análise do texto de um documento pode não deixar explícita uma classe importante para o sistema. Em linguagem natural, as variações linguísticas e as formas de expressar uma mesma ideia são bastante numerosas.

Análise de Casos de Uso:

* Essa técnica é também chamada de identificação dirigida por casos de uso, sendo um caso particular da técnica de Abbott.
* Técnica preconizada pelo Processo Unificado.
* O MCU é utilizado como ponto de partida.
  + Premissa: um caso de uso corresponde a um comportamento específico do software. Esse comportamento somente pode ser produzido por objetos que compõem o sistema.
  + Com base nisso, o modelador aplica a técnica de análise dos casos de uso para identificar as classes necessárias à produção do comportamento que está documentado na descrição do caso de uso.
* Procedimento de aplicação:
  + O modelador estuda a descrição textual de cada caso de uso para identificar classes candidatas.
  + Para cada caso de uso, seu texto (fluxos principal, alternativos e de exceção, pós-condições e pré-condições, etc.) é analisado.
  + Na análise de certo caso de uso, o modelador tenta identificar classes que possam fornecer o comportamento do mesmo.
  + Na medida em que os casos de uso são analisados um a um, as classes do software são identificadas.
  + Quando todos os casos de uso tiverem sido analisados, todas as classes (ou pelo menos a grande maioria delas) terão sido identificadas.

Categorização BCE:

* Na aplicação deste procedimento, podemos utilizar a categorização BCE, na qual os objetos são agrupados de acordo com o tipo de responsabilidade a eles atribuída.
  + objetos de entidade: usualmente objetos do domínio do problema
  + objetos de fronteira: atores interagem com esses objetos
  + objetos de controle: servem como intermediários entre objetos de fronteira e de entidade, definindo o comportamento de um caso de uso específico.
* Possui correspondência com o padrão model-view-controller (MVC)
* Estereótipos na UML: «boundary», «entity», «control»
* A importância dessa categorização está relacionada à capacidade de adaptação a eventuais mudanças. Se cada objeto tem atribuições específicas dentro do sistema, mudanças podem ser menos complexas e mais localizadas.
* Uma modificação em uma parte do sistema tem menos possibilidades de resultar em mudanças em outras partes.

Objetos de Entidade:

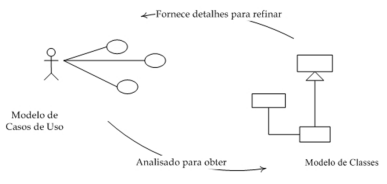
* Repositório para informações e as regras de negócio manipuladas pelo sistema. Representam conceitos do domínio do negócio.
* Normalmente armazenam informações persistentes. Várias instâncias da mesma entidade existindo no sistema. Participam de vários casos de uso e têm ciclo de vida longo.
* Exemplo: Um objeto Pedido participa 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 Fronteira:

* Realizam a comunicação do sistema com os atores.
* Traduzem os eventos gerados por um ator em eventos relevantes ao sistema => eventos de sistema.
* Também são responsáveis por apresentar os resultados de uma interação dos objetos em algo inteligível pelo ator.
* Existem para que o sistema se comunique com o mundo exterior.
* Por conseqüência, são altamente dependentes do ambiente.
* Há dois tipos principais de objetos de fronteira:
  + Os que se comunicam com o usuário (atores humanos): relatórios, páginas HTML, interfaces gráfica desktop, etc.
  + Os que se comunicam com atores não-humanos (outros sistemas ou dispositivos): protocolos 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 de sistema ocorre. Eles realizam o controle do processamento
* Agem como gerentes (coordenadores, controladores) dos outros objetos para a realização de um caso de uso.
* Traduzem eventos de sistema em operações que devem ser realizadas pelos demais objetos.



**Modelo de Iterações**

Esse modelo representa mensagens trocadas entre objetos para a execução de cenários dos casos de uso do sistema.

A construção dos diagramas de interação é uma consolidação do entendimento dos aspectos dinâmicos do sistema.

A modelagem de interações é uma parte da modelagem dinâmica de um software OO.

Diagramas de interação representam como o sistema age internamente para que um ator atinja seu objetivo na realização de um caso de uso. A modelagem de um software OO normalmente contém diversos diagramas de interação. O conjunto de todos os diagramas de interação de um sistema constitui o seu modelo de interações.

O conceito básico da interação entre objetos é a mensagem.

* Um sistema OO é uma rede de objetos que trocam mensagens.
  + Funcionalidades são realizadas pelos objetos, que só podem interagir através de mensagens.
  + Um objeto envia uma mensagem para outro objeto quando o primeiro deseja que o segundo realize alguma tarefa.
* O fato de um objeto “precisar de ajuda” indica a necessidade de este enviar mensagens.

Na construção de diagramas de interação, mensagens de um objeto a outro implicam em operações que classes devem ter.

Uma mensagem representa a requisição de um objeto remetente a um objeto receptor para que este último execute alguma operação definida para sua classe. Essa mensagem deve conter informação suficiente para que a operação do objeto receptor possa ser executada.

