Andrey Ricardo Pimentel

Projeto de Software Usando a UML

Apostila para Curso de Projeto de Sistemas Orientado a Objetos Usando a UML.

Setembro de 2015

Sumário

AULA 1 PROCESSO DE SOFTWARE....5
1.1. Apresentação....5
1.2. Introdução....5
1.3. UML....5
1.4. Crise do software....6
1.5. Processo e Notação....6
1.6. O poder da tecnologia de Objetos....6
1.7. Análise e projeto....7
1.8. Desenvolvimento Iterativo e Incremental....7
1.9. Processo Unificado....7
1.10. Atividade....9
AULA 2 LEVANTAMENTO DE REQUISITOS....10
2.1. Apresentação....10
2.2. Levantamento de Requisitos....10
2.3. Atividade....11
AULA 3 CASOS DE USO....13
3.1. Apresentação....13
3.2. Comportamento do Sistema....13
3.3. Atores....13
3.4. Casos de Uso....14
3.5. DIAGRAMAS DE CASO DE USO....15
3.6. Atividade....16
AULA 4 ESPECIFICAÇÃO DE CASOS DE USO....17
4.1. Apresentação....17
4.2. A Especificação de um caso de Uso....17
4.3. Atividade....18
AULA 5 RELACIONAMENTOS ENTRE CASOS DE USO....19
5.1. Apresentação....19
5.2. Relacionamentos entre casos de uso....19
5.3. Relacionamento <<include>>....19
5.4. Relacionamento <<extend>>....19
5.5. Generalizações....20
5.6. Atividade....21
AULA 6 IDENTIFICAÇÃO DE CLASSES USANDO O MVC....22
6.1. Apresentação....22
6.2. Descobrindo Classes....22
6.3. Classes Entidade....22
6.4. Classes Limite....22
6.5. Classes Controle....23
6.6. Objetos e Classes no problema do Sistema de Matrícula (MATRI)....23
6.7. Atividade....24
AULA 7 IDENTIFICAÇÃO DAS RESPONSABILIDADES DAS CLASSES 25
7.1. Apresentação....25
7.2. Cartões CRC....25
7.3. Atividade....26
AULA 8 IDENTIFICAÇÃO DE ATRIBUTOS E OPERAÇÕES DE UMA CLASSE....27
8.1. Apresentação....27

8.2. O que é uma operação?....27
8.3. O que é um Atributo?....29
8.4. Encapsulamento....29
8.5. Atividade....31
AULA 9 ASSOCIAÇÕES ENTRE CLASSES....32
9.1. Apresentação....32
9.2. Associações....32
9.3. Multiplicidade para Associações....33
9.4. Atividade....34
AULA 10 AGREGAÇÕES E ASSOCIAÇÕES....36
10.1. Apresentação....36
10.2. Agregação....36
10.3. Associações Reflexivas....37
10.4. Classes de Ligação....38
10.5. Encontrando Associações e Agregações....39
10.6. Atividade....39
AULA 11 HERANÇA....40
11.1. Apresentação....40
11.2. Generalização....40
11.3. Herança Múltipla....42
11.4. Atividade....43
AULA 12 INTERFACES E PACOTES....44
12.1. Apresentação....44
12.2. Interfaces....44
12.3. Pacotes....45
12.4. Atividade....47
AULA 13 MODELO DE CLASSES, IMPLEMENTAÇÃO E MODELO RELACIONAL....48
13.1. Apresentação....48
13.2. Mapeando diagramas de Classe para código em java....48
13.3. Mapeando Classes para Modelo RElacional....50
13.4. Atividade....52
AULA 14 DIAGRAMAS DE INTERAÇÃO....53
14.1. Apresentação....53
14.2. Diagramas de interação....53
14.3. Atividade....55
AULA 15 DIAGRAMAS DE SEQÜÊNCIA....56
15.1. Apresentação....56
15.2. DESENVOLVIMENTO....56
15.3. Atividade....58
AULA 16 DIAGRAMAS DE ESTADOS....59
16.1. Apresentação....59
16.2. Diagramas de Estados:....59
16.3. Atividade....61
AULA 17 DIAGRAMAS DE COMPONENTES E DE IMPLANTAÇÃO....62
17.1. Apresentação....62
17.2. Diagramas de Componentes....62
17.3. Diagramas de Implantação....63
17.4. Atividade....64
17.5. BIBLIOGRAFIA BÁSICA....64

AULA 1 PROCESSO DE SOFTWARE

1.1. APRESENTAÇÃO

Nesta aula serão apresentados e discutidos os conceitos de processo de desenvolvimento de software, processo e notação, crise de software, análise e projeto, desenvolvimento iterativo e incremental e do Processo Unificado.

1.2. INTRODUÇÃO

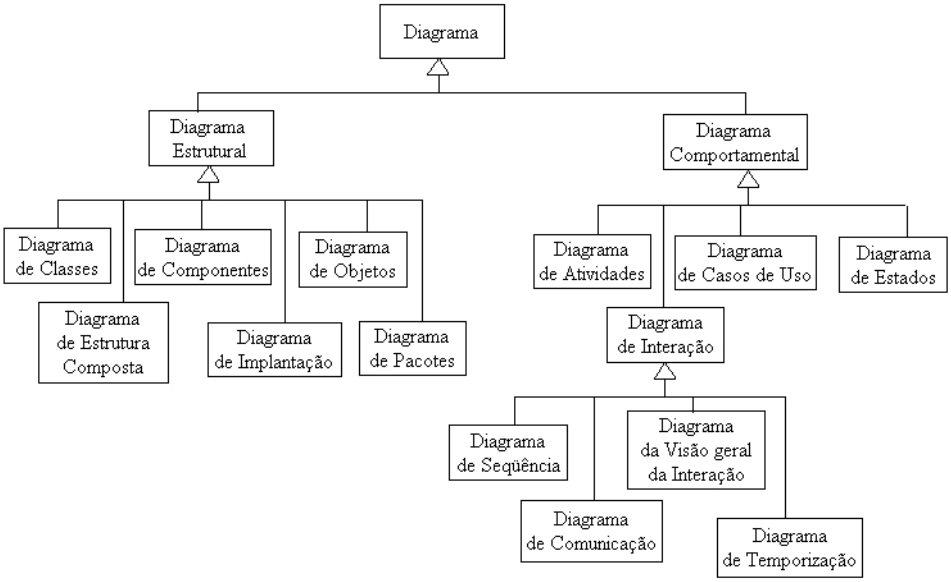
O desenvolvimento orientado a objetos começou em 1967 com a linguagem Simula-67 e desde então surgiram linguagens orientadas a objetos como Smalltalk e C++ entre outras. Nos anos 80 começaram a surgir metodologias de desenvolvimento orientadas a objetos para tirar vantagens deste paradigma. Entre 1989 e 1994 surgiram quase 50 métodos de desenvolvimento orientados a objetos, fenômeno que foi chamado a guerra dos métodos.

Entre as mais importantes estavam: o método de G. Booch, a *Object Modeling Technique* (OMT) de J. Rumbaugh, o método de Shlaer e Mellor e o método *Objectory* de I. Jacobson. I. Jacobson introduziu a modelagem de casos de uso em 1987 e criou o primeiro processo de desenvolvimento de software que utiliza casos de uso, chamado *Objectory*.

Cada método possuía uma notação própria, o que gerou uma infinidade de tipos de diagramas e notações. Isto causava problemas de comunicação, treinamento de pessoal e portabilidade. Um esforço de unificação começou em 1994 quando J. Rumbaugh e, logo após, I. Jacobson, juntaram-se a G. Booch, na empresa *Rational Software Corporation*. O primeiro grande resultado desse esforço foi a criação da *Unified Modeling Language* (UML), apresentada, na sua versão 1.0, em 1997.

1.3. UML

A UML é uma linguagem criada para visualizar, especificar, construir e documentar os artefatos de um sistema de software. A UML é adotada, desde 1997, como padrão internacional pelo OMG (Object Management Group). A UML provê um conjunto de diagramas e seus componentes, todos com notação e comportamento (semântica) bem definidos. A UML descreve 13 diagramas que são apresentados na figura abaixo.



1.4. CRISE DO SOFTWARE

No começo os sistemas computacionais, tinham um custo de hardware muitas vezes maior que o custo do software. O software tinha um caráter "descartável". Com a diminuição dos custos do hardware e o aumento da complexidade do software, o custo do software começou a ser notado. Com isso o software deixou de ser descartável. Aumentaram as preocupações com manutenção e evolução dos softwares das empresas. Qualidade de software passou a ser fundamental. Fazer software deixou de ser arte para ser engenharia. Surgiram processos de desenvolvimento

1.5. PROCESSO E NOTAÇÃO

Processo de desenvolvimento de software pode ser definido como uma seqüência de etapas para a construção do software. Por exemplo: Especificação de requisitos, análise, projeto, implementação, testes e implantação. Cada etapa tem objetivos bem definidos e gera um conjunto de artefatos. Artefatos são os produtos gerados em cada etapa. Por exemplo: uma etapa de análise pode gerar os diagramas de caso de uso. Milestones são pontos do processo onde os artefatos da etapa devem ser sincronizados e validados. Por exemplo: o diagrama de classe deve ser sincronizado com o diagrama de seqüência. Notação é a linguagem, gráfica ou textual, usada na elaboração dos artefatos. UML é uma linguagem de modelagem.

1.6. O PODER DA TECNOLOGIA DE OBJETOS

Amento da Coesão entre os módulo e diminuição do Acoplamento

Aumento da reutilização de código

Facilita a criação de bibliotecas de classe da empresa

Redução do tempo de desenvolvimento dos projetos

Redução do número de linhas de código dos projetos

1.7. ANÁLISE E PROJETO

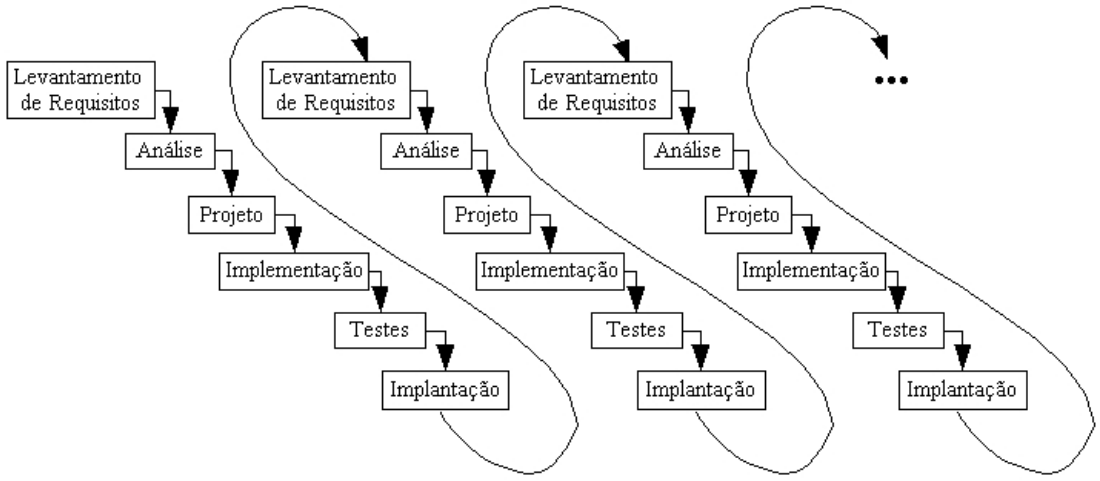
Análise é a descrição do problema a ser implementado Na análise você descreve os objetos e classes que existem no problema, suas relações e como eles se comportam durante os eventos que existem no problema a ser implementado.

Projeto é a descrição da solução adotada na implementação. No projeto você descreve como os objetos que você descreveu na análise vão se comportar na solução que você criou. Também são criados novos objetos, relações e comportamentos para facilitar e melhorar o sistema que vai ser construído. No projeto, também é descrita a arquitetura do sistema, sua base da dados, entre outros. O limite entre análise e projeto não é muito bem definido.

1.8. DESENVOLVIMENTO ITERATIVO E INCREMENTAL

O desenvolvimento de sistemas seguindo o Processo Unificado é:

- Iterativo e incremental
- Guiado por casos de uso (*use cases*)
- Baseado na arquitetura do sistema



1.9. PROCESSO UNIFICADO

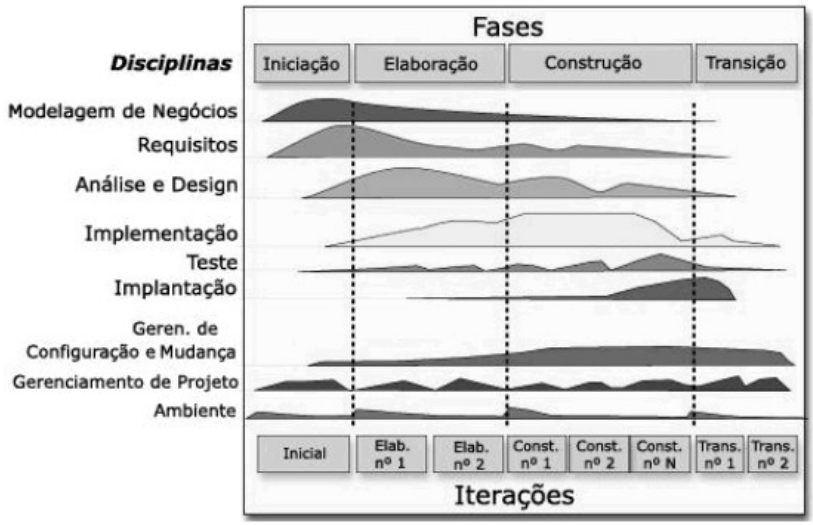
Outro grande resultado deste esforço de unificação de metodologias foi a criação, pela Rational, de um processo de desenvolvimento que usa a UML em seus modelos, chamado Processo Unificado. O Processo Unificado evoluiu do processo Rational Objectory, sendo inicialmente chamado de Rational Unified Process (RUP). O Processo Unificado, apesar de não ser um padrão, é amplamente adotado, sendo considerado como um modelo de processo de desenvolvimento de software orientado a objetos.

O ciclo de vida de um sistema consiste de quatro fases, divididas em iterações:

- Concepção (define o escopo do projeto)
- Elaboração (define os requisitos e a arquitetura)
- Construção (desenvolve o sistema)

- Transição (implanta o sistema)

Cada fase é dividida em iterações e cada iteração
- é planejada
- realiza uma seqüência de atividades (de elicitação de requisitos, análise e projeto, implementação, etc.) distintas
- geralmente resulta em uma versão executável do sistema
- é avaliada segundo critérios de sucesso previamente definidos



O Processo Unificado é guiado por casos de uso
- Os casos de uso não servem apenas para definir os requisitos do sistema
- Várias atividades do Processo Unificado são guiadas pelos casos de uso:
- planejamento das iterações
- criação e validação do modelo de projeto
- planejamento da integração do sistema
- definição dos casos de teste

O Processo Unificado é baseado na arquitetura do sistema
- Arquitetura é a visão geral do sistema em termos dos seus subsistemas e como estes se relacionam
- A arquitetura é prototipada e definida logo nas primeiras iterações
- O desenvolvimento consiste em complementar a arquitetura
- A arquitetura serve para definir a organização da equipe de desenvolvimento e identificar oportunidades de reuso

Fluxos de atividades
- Atividades
- passos
- entradas e saídas
- guias (de ferramentas ou não), *templates*
- Responsáveis (papel e perfil, não pessoa)
- Artefatos

1.10. ATIVIDADE

1) Qual a diferença entre processo e notação?
2) Qual a diferença entre análise e projeto de sistemas de software?
3) Cite algumas características da orientação a objetos.
4) Quais são as características do processo unificado?
5) Quais são e o que é feito em cada uma das fases do processo unificado?
6) O que é arquitetura do sistema?

AULA 2 LEVANTAMENTO DE REQUISITOS

2.1. APRESENTAÇÃO

Nesta aula serão apresentados e discutidos os conceitos de concepção e especificação de um projeto de um sistema de software.

2.2. LEVANTAMENTO DE REQUISITOS

A atividade de levantamento de requisitos corresponde à etapa de compreensão do problema. O levantamento de requisitos fornece o mecanismo adequado para entender o que o cliente deseja, analisando suas necessidade, e determinando se elas são possíveis.

Dificuldades no levantamento de requisitos:

- Problemas de escopo: o limite do sistema é mal definido ou o cliente especifica detalhes técnicos que atrapalham a especificação.
- Problemas de entendimento: Os clientes ou usuários não estão certos do que é necessário para o sistema, não têm conhecimento sobre o domínio do problema, não têm conhecimento sobre as limitações técnicas ou omitem detalhes técnicos.
- Problemas de volatilidade: Os requisitos mudam ao longo do tempo.

Durante o levantamento de requisitos a equipe tenta entender o domínio que deve ser automatizado pelo sistema de software. O levantamento de requisitos é um estudo exploratório das necessidades dos clientes, usuários e *stakeholders* (pessoas que são afetadas pelo sistema).

O produto do levantamento de requisitos é o documento de requisitos. Neste documento, os requisitos são categorizados em: requisitos funcionais, requisitos não funcionais e requisitos normativos.

Os requisitos funcionais representam as necessidades que o sistema deve prover. Por exemplo:

- “O sistema deve permitir que o professor lance as notas de um aluno”,
- “O sistema deve permitir que o cliente se cadastre para receber emails”,
- “O sistema deve permitir que o gerente de vendas visualize o relatório de vendas por região”.

Os requisitos não funcionais representam características de qualidade que o sistema deve ter e que não estão relacionadas com suas funcionalidades. Alguns tipos de requisitos funcionais são:

- Confiabilidade: tempo médio entre falhas, recuperação de falhas ou número de erros

por milhares de linhas de código.

- Desempenho: tempo de resposta esperado para cada funcionalidade do sistema.
- Portabilidade: restrições sobre as plataformas de hardware ou software nas quais o sistema será implementado e o grau de portabilidade para outras plataformas.
- Segurança: limitações sobre segurança do sistema em relação a acessos não autorizados.
- Usabilidade: requisitos sobre facilidade de uso, idiomas, acessibilidades especiais, necessidade ou não de treinamento.

Os requisitos normativos representam restrições impostas sobre o desenvolvimento do sistema como: adequações a custos, prazos, plataforma, aspectos legais, além de regras de negócio e políticas de funcionamento.

O documento de requisitos serve como um termo de acordo entre o cliente e a equipe de desenvolvedores. Ele servirá de base para as atividades de desenvolvimento e para validações posteriores. O documento de requisitos estabelece o escopo do sistema, ou seja, o que faz parte do sistema e o que não faz parte do sistema.

2.3. ATIVIDADE

1)Faça uma especificação de requisitos para o sistema descrito abaixo:

Sistema de Matrícula em cursos de Universidade

O processo de designação de professores para cursos e a matrícula de estudantes é uma experiência frustrante e que consome muito tempo.
Depois que os professores da Universidade decidem em quais cursos eles vão lecionar no semestre, a Secretaria alimenta essa informação no sistema em computador. Um relatório *batch* é impresso para os professores indicando em quais cursos eles lecionarão. Um catálogo de cursos é impresso e distribuído para os estudantes.
Atualmente, os alunos preenchem formulários de matrícula que indicam suas escolhas de cursos e retornam os formulários para a Secretaria. A carga típica de um estudante é de quatro cursos. Os funcionários da Secretaria alimentam os dados dos formulários dos estudantes no sistema de computador no *mainframe.* Uma vez que o currículo para o semestre foi alimentado no sistema, um *job batch* é executado durante a noite para inscrever os estudantes nos cursos. Na maioria das vezes os estudantes obtêm sua primeira escolha; entretanto, naqueles casos em que há um conflito, a Secretaria conversa com cada estudante para obter escolhas adicionais. Quando todos os estudantes tiverem sido inscritos com sucesso nos cursos, um relatório dos currículos dos estudantes é encaminhado para eles para verificação. A maioria dos registros dos estudantes são processados durante uma semana, mas alguns casos excepcionais demoram até duas semanas para serem solucionados. Quando o período inicial de matrícula é concluído, os professores recebem uma lista de estudantes para cada curso que devem lecionar.

O novo sistema vai permitir, no início de cada semestre, que os estudantes possam solicitar um catálogo de cursos contendo uma lista dos cursos oferecidos no semestre. Informações sobre cada curso, tais como professor, departamento e pré-requisitos estarão incluídas para ajudar os estudantes a tomarem suas decisões.
O novo sistema vai permitir aos estudantes selecionarem quatro dos cursos oferecidos para o semestre. Além disso, cada estudante indicará duas escolhas alternativas para caso um curso oferecido seja cancelado ou não tenha vagas suficientes. Nenhum curso poderá ter mais de dez ou menos do que três alunos. Um curso com menos de três alunos será cancelado. Uma vez concluído o processo de matrícula de um estudante, o

sistema de matrícula envia informação para o sistema de faturamento de forma que o aluno possa receber as faturas para o semestre.

Professores devem poder acessar o sistema *online* para indicar quais cursos eles lecionarão e para ver que estudantes se inscreveram em suas ofertas de curso. Para cada semestre, há um período de tempo no qual os estudantes podem mudar sua grade horária. Os estudantes devem poder acessar o sistema durante este período para adicionar ou retirar cursos.

AULA 3 CASOS DE USO

3.1. APRESENTAÇÃO

Nesta aula serão discutidos de comportamento do sistema, como definir casos de uso e atores e como utilizar o diagrama de casos de uso para mostrar os atores, os casos de uso e suas interações.

3.2. COMPORTAMENTO DO SISTEMA

O comportamento do sistema em desenvolvimento, que é o que funcionalmente deve ser fornecido pelo sistema, é documentado em um **Modelo de Caso de Uso** que ilustra as funções pretendidas do sistema (casos de uso), suas vizinhanças (atores) e relacionamentos entre os casos de uso e atores (diagramas de casos de uso). O papel mais importante de um modelo de caso de uso é o de comunicação. Ele provê um veículo usado pelo cliente ou usuários finais e os desenvolvedores, para discutir a funcionalidade e o comportamento do sistema.

O modelo de caso de uso é iniciado na **Fase de Concepção** com a identificação dos atores e casos de uso principais do sistema. O modelo é então amadurecido na Fase de Elaboração - informação mais detalhada é adicionada aos casos de uso identificados, e casos de uso adicionais são introduzidos à medida que forem necessários.

3.3. ATORES

Atores não são parte do sistema - eles representam algo ou alguém que deve interagir com o sistema. Um ator pode:

- Somente fornecer informações para o sistema
- Somente receber informações do sistema
- Fornecer e receber informações para e do sistema

Tipicamente, estes atores são encontrados na definição do problema e em conversas com clientes e especialistas no domínio do problema. As seguintes perguntas podem ser usadas para auxiliar na identificação dos atores de um sistema:

- Quem está interessado numa certa necessidade?
- Onde na organização o sistema é usado?
- Quem se beneficiará do uso do sistema?
- Quem suprirá o sistema com esta informação, usará esta informação e removerá esta informação?
- Quem dará o suporte e manterá o sistema?
- O sistema usa um recurso externo?
- Uma pessoa desempenha diferentes papéis?
- Várias pessoas desempenham o mesmo papel?
- O sistema interage com um sistema legado?

Na **UML,** um ator é representado como um homem palito, como mostrado na figura abaixo.

Notação UML para um Ator

O que constitui um bom ator?
Devemos tomar cuidado quando identificamos um ator para o sistema. Esta identificação é feita de uma maneira iterativa - a primeira lista de atores para um sistema raramente constitui a lista final. Por exemplo: um novo estudante é um ator diferente de um estudante que está retornando? Suponha que você inicialmente responda afirmativamente a esta questão. O próximo passo é identificar como o ator interage com o sistema. Se o novo estudante usa o sistema de forma diferente de um estudante que retorna, eles são atores diferentes. Se eles usam o sistema da mesma forma, eles são o mesmo ator.

Atores no Sistema de Matrícula - MATRI
Respondendo às perguntas anteriores, identificaremos diversos atores. Um deles é Professor.

Documentação de Atores
Uma breve descrição para cada ator deveria ser adicionada ao modelo. A descrição deveria identificar o papel que o ator desempenha na interação com o sistema. Exemplo: Professor - uma pessoa que é certificada para lecionar na universidade

3.4. CASOS DE USO

Casos de Uso modelam um diálogo entre um ator e o sistema. Eles representam a funcionalidade fornecida pelo sistema; isto é, que capacidades serão providas para o ator pelo sistema. A coleção de casos de uso para um sistema constitui todos os caminhos definidos nos quais o sistema pode ser usado. A definição formal para um caso de uso é:

- Um caso de uso é uma seqüência de transações executadas por um sistema, que produz um resultado mensurável de valores para um ator em particular.

As seguintes perguntas devem ser usadas para auxiliar na identificação dos casos de uso para um sistema:

- Quais são as tarefas de cada ator?
- Algum ator criará, armazenará, mudará, removerá ou lerá informação do sistema?
- Que casos de uso criarão, armazenarão, mudarão, removerão ou lerão esta informação?
- Algum ator precisará informar o sistema a respeito de mudanças externas repentinas?
- Algum ator necessita ser informado a respeito de certas ocorrências no sistema?
- Que casos de uso suportarão ou manterão o sistema?
- Todas as necessidades funcionais podem ser executadas pelos dos casos de uso?

Na UML, um caso de uso é representado como uma figura oval, como mostrado na figura abaixo.

Notação para caso de uso

O que Constitui um Bom Caso de Uso?

Ao longo dos anos tem havido muita discussão sobre o que é um bom caso de uso. Um problema freqüentemente encontrado é o nível de detalhe dos casos de uso. Isto é, quão grande (ou quão pequeno) ele deveria ser? Não há uma resposta certa. Uma boa regra aplicável é:

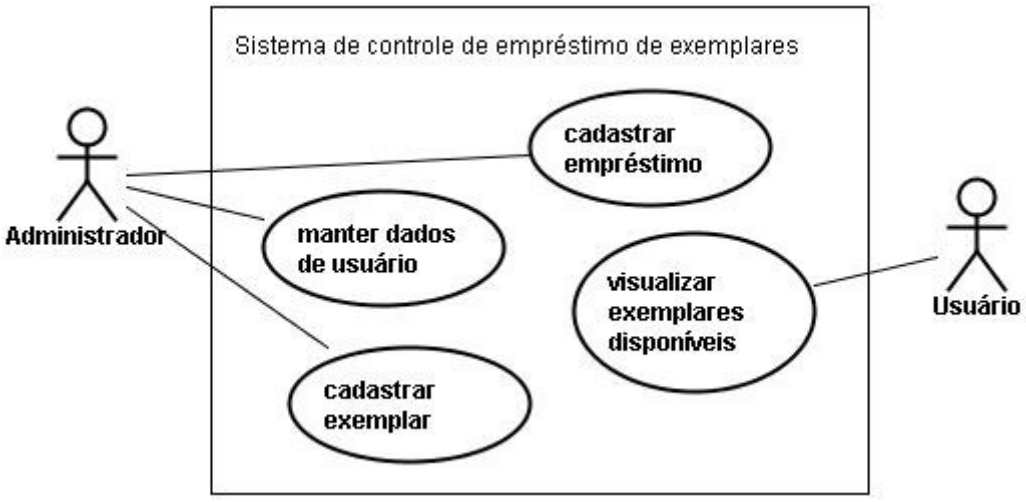
Um caso de uso tipicamente representa uma peça maior de funcionalidade que está completa do início ao fim. Um caso de uso deve fornecer algo de valor para um ator.

Outro problema é como empacotar funcionalidade que é diferente mas que aparentemente deveria permanecer junta. Por exemplo, a Secretaria deve incluir cursos, eliminar cursos e modificar cursos. Três casos de uso ou um único? Aqui novamente, deveria ser feito um caso de uso - a manutenção de currículo, já que a funcionalidade é iniciada pelo mesmo ator (a Secretaria) e trata com as mesmas entidades no sistema (o currículo).

3.5. DIAGRAMAS DE CASO DE USO

Um diagrama de caso de uso é uma visão gráfica de alguns ou todos os atores, casos de usos e seus relacionamentos identificados para um sistema. Cada sistema normalmente tem um Diagrama de Caso de Uso principal, o qual é uma representação da fronteira do sistema (atores) e a maior funcionalidade fornecida pelo sistema (casos de uso). Outros diagramas de casos de uso podem ser criados quando necessário. Alguns exemplos são:

- um diagrama que mostre todos os casos de uso para um ator selecionado;
- um diagrama mostrando todos os casos de uso a serem implementados em uma iteração;
- um diagrama mostrando um caso de uso e todos os seus relacionamentos com outros casos de uso e atores.



Exemplo de diagrama de casos de uso

Casos de Uso no Sistema Matrícula - MATRI

As seguintes necessidades devem ser tratadas pelo sistema:

- O ator estudante precisa usar o sistema para registrar-se em cursos
- Depois que o processo de seleção estiver completo, o sistema de Faturamento deve ser suprido com informações de fatura
- O ator Professor precisa usar o sistema para selecionar os cursos para lecionar por um semestre e deve estar habilitado a receber uma lista de cursos do sistema
- A Secretaria é responsável pela geração do catálogo de curso para um semestre e, pela

manutenção de todas as informações a respeito do currículo, dos estudantes e dos professores.

Baseado nestas necessidades, diversos casos de usos podem ser identificados, como o "Matricula em um curso".

3.6. ATIVIDADE

Baseando-se na descrição dos sistema de matrículas (MATRI), descrito acima, desenvolva as seguintes atividades:

1)Encontre os casos de uso, os atores, e os relacionamentos entre os casos de uso e atores (coloque tudo no diagrama de casos de uso)

2)Faça uma breve descrição para os atores e os casos de uso que forem identificados (máximo 2 frases)

AULA 4 ESPECIFICAÇÃO DE CASOS DE USO

4.1. APRESENTAÇÃO

Nesta aula será apresenta e discutida a especificação dos caso de uso

4.2. A ESPECIFICAÇÃO DE UM CASO DE USO

Cada caso de uso também é documentado com uma Especificação de Caso de Uso. Uma Especificação dos casos de uso é um documento que tem por objeto descrever o comportamento do casos de uso. Este comportamento é descrito basicamente através do fluxo principal de eventos.

O fluxo de eventos para um caso de uso é uma descrição dos eventos necessários para atingir o comportamento esperado do caso de uso. O fluxo de eventos é escrito em termos do que o sistema deveria fazer, não como o sistema o faz. Isto é, é escrito na linguagem do domínio, não em termos de implementação. Especificação de Caso de Uso deveria incluir:

- Quando e como o caso de uso inicia e termina
- Que interação o caso de uso tem com os atores
- Que dados são necessários para o caso de uso
- A seqüência normal de eventos para o caso de uso

A Especificação de Caso de Uso é criada tipicamente na Fase de Elaboração numa maneira iterativa. Inicialmente, somente uma breve descrição dos passos necessários para executar o fluxo normal do caso de uso (isto é, que funcionalidade é fornecida pelo caso de uso) é escrita. À medida que a análise progride, os passos são preenchidos a fim de se adicionar mais detalhes. Finalmente, os fluxos de exceção são adicionados ao caso de uso.

Cada projeto deveria usar um *template* padrão para a criação da documentação do fluxo de eventos. É útil o seguinte *template*:

1Nome do Caso de Uso
2Breve descrição
3Fluxo de eventos
3.1Fluxo básico
3.1.1< Primeiro evento do fluxo básico>
3.1.2< Primeiro evento do fluxo básico>
3.2Fluxos alternativos
3.2.1< Primeiro fluxo alternativo>
3.2.2< Outro fluxo alternativo>
4Requisitos Especiais
4.1< primeiro requisito especial >
5Pré-condições
5.1< pré-condição um >
6Pós-condições
6.1< pós-condição um >
7Pontos de Extensão

Uma amostra do documento de Fluxo de Eventos para o Caso de Uso Selecionar Cursos para

Lecionar é mostrada a seguir:

Especificação do Caso de Uso Cadastrar Usuário

1 Nome do Caso de Uso
Cadastrar Usuário
2 Breve descrição
Cadastra um usuário no sistema, com todos os dados, inclusive uma fotografia.
3 Fluxo de eventos
3.1Fluxo básico
3.1.1 O caso de uso começa quando o ator seleciona a opção cadastrar usuário. O sistema mostra a tela de cadastro de usuário com os seguintes campos em branco: “nome”, "endereço", “RG”, “CPF”, “telefone”, “email”.
3.1.2 O ator preenche os campos e seleciona a opção cadastrar. O sistema mostra uma tela para fazer o upload da fotografia.
3.1.3 O ator indica o nome e o caminho do arquivo com a sua fotografia e seleciona importar. O sistema mostra uma tela com todos os dados e a fotografia do cliente.
3.1.4 O ator seleciona a opção cadastrar e o sistema cadastra o novo usuário, mostra uma mensagem de sucesso na operação e mostra a tela inicial do sistema.
3.2 Fluxos alternativos
3.2.1< Primeiro fluxo alternativo>
Caso o ator não tenha preenchido um dos campos dos dados do usuário, o sistema avisa o erro e retorna para a tela de cadastro dos dados do usuário com os dados já preenchidos.
3.2.2< segundo fluxo alternativo>
Caso o usuário não importe um arquivo com a foto, o sistema mostra a tela de confirmação sem a fotografia e uma mensagem de aviso.
3.2.3< Terceiro fluxo alternativo>
Caso o ator tenha cadastrado um usuário com o mesmo CPF, o sistema avisa o erro e retorna para a tela de cadastro dos dados do usuário com os dados já preenchidos.
4 Requisitos Especiais
Não se aplica
5 Pré-condições
< pré-condição um >
O ator deve estar logado no sistema. Nenhum outro usuário com o mesmo CPF dever ter sido cadastrado.
6 Pós-condições
< pós-condição um >
Um usuário novo é cadastrado.
7 Pontos de Extensão
Não se aplica.

4.3. ATIVIDADE

1)Baseando-se na descrição dos sistema de matrículas (MATRI), descrito nas unidades anteriores, selecione 3 dos casos de uso identificados e faça a especificação de casos de uso para cada um deles

AULA 5 RELACIONAMENTOS ENTRE CASOS DE USO

5.1. APRESENTAÇÃO

Nesta aula será apresentado e discutido como utilizar o diagrama de casos de uso para mostrar os atores, os casos de uso e suas interações.

5.2. RELACIONAMENTOS ENTRE CASOS DE USO

Um relacionamento de associação pode existir entre um ator e um caso de uso. Esse tipo de associação é normalmente chamado como uma *Associação de Comunicação*, desde que ela represente uma comunicação entre um ator e um caso de uso. Uma associação é representada como uma linha que liga os elementos a serem relacionados. A navegação em somente uma direção pode ser representada pela adição de uma seta que indica a direção na linha da associação. Não pode existir no modelo um caso de uso iniciado por dois atores. Existem somente 3 tipos de relacionamentos entre os casos de uso: <<include>>, <<extend>> e a generalização.

5.3. RELACIONAMENTO <<INCLUDE>>

Muitos casos de uso podem compartilhar pedaços de pequenas funcionalidades. Esta funcionalidade é colocada em separado em outro caso de uso ao invés de ser documentada em cada caso de uso que precisa dela. Relacionamentos de <<include>> são criados entre um novo caso de uso e qualquer outro caso de uso que utilize esta funcionalidade. Por exemplo, os casos de uso remover cliente e alterar cliente precisam pesquisar o cliente a ser removido ou alterado. Essa funcionalidade pode ser colocada em um caso de uso chamado de “pesquisar cliente”, o qual então é incluído por outros casos de uso quando necessário.

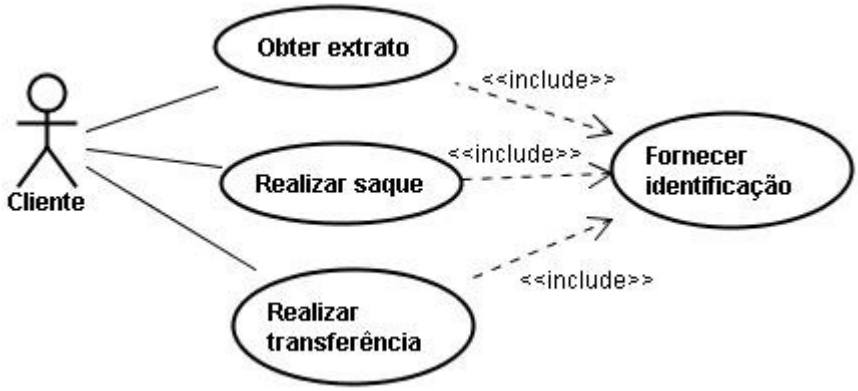


Figura relacionamento <<include>>

5.4. RELACIONAMENTO <<EXTEND>>

Um relacionamento de "extend" é usado para mostrar: comportamento opcional, comportamento que somente é executado sobre determinadas condições, como o disparo de

um alarme, muitos diferentes caminhos que podem ser executados de acordo com uma seleção feita por um ator. Por exemplo, um caso de uso que monitora o fluxo de pacotes em uma esteira de transporte pode acionar um caso de uso de Disparo de Alarme se os pacotes empilharem. Até este momento, nenhum <<extend>> foi identificado para o Sistema de Matrícula (MATRI).

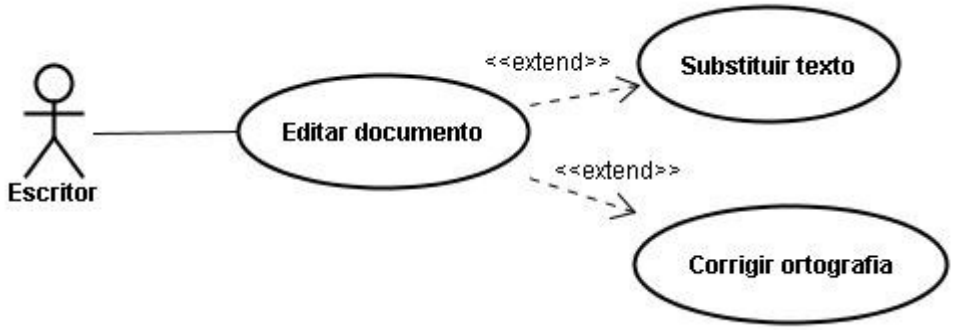


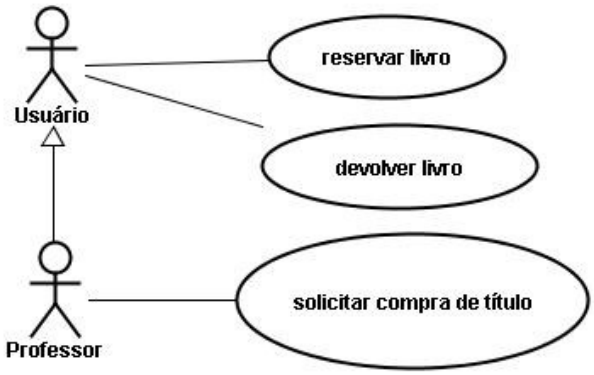
Figura relacionamento <<extend>>

Um relacionamento extend de um caso de uso A para um caso de uso B indica que o caso de uso B pode ser aumentado (de acordo com condições especificadas na extensão) por um comportamanto especificado pelo caso de uso A. O comportamento é inserido no local definido pelo ponto de extensão em B o qual é referenciado pelo relacionamento extend. No caso de uso A, o comportamento a ser inserido deve ser marcado com um “rótulo”.

5.5. GENERALIZAÇÕES

Uma generalização entre um caso de uso C e um caso de uso D indica que C é uma especialização de D. Este relacionamento é representado por uma seta de generalização partindo de D para C.

Pode ser representado, também, um tipo de relacionamento entre atores. Este relacionamento é o de generalização. Uma generalização de um ator A para um ator B indica que A pode se comunicar com os mesmos casos de uso que B.



Siga a seguinte regra:

- Utilize <<extend>> quando estiver descrevendo uma variação do comportamento normal de um caso de uso;
- Utilize <<include>> para permitir a reutilização de um determinado comportamento de um caso de uso por outros casos de uso.

5.6. ATIVIDADE

Baseando-se na descrição dos sistema de matrículas (MATRI), descrito na unidade anterior, desenvolva as seguintes atividades:

1)Encontre os casos de uso, os atores, e os relacionamentos entre os casos de uso e atores (coloque tudo no diagrama de casos de uso)

AULA 6 IDENTIFICAÇÃO DE CLASSES USANDO O MVC

6.1. APRESENTAÇÃO

Nesta aula será visto como identificar Classes de um sistema usando o Padrão Modelo-Visão-Controlador.

6.2. DESCOBRINDO CLASSES

Não existe um livro de receitas que ensine como descobrir classes. O RUP (Rational Unified Process) sugere que as classes sejam descobertas no desenvolvimento do sistema, procurando as classes de limite, controle e entidade. Estes três estereótipos ajustam-se, ponto de vista *model-view-controler* e permitem ao analista particionar o sistema separando a visão do domínio do controle necessário pelo sistema.

Tendo como base, que a análise e o projeto do processo são interativos, a lista de classes irá mudar ao longo do tempo. O conjunto inicial de classes provavelmente não será o conjunto de classes que serão efetivamente implementadas. Para nós, o termo candidato para uma classe é freqüentemente usado para descrever o primeiro conjunto de classes descobertas para o sistema.

6.3. CLASSES ENTIDADE

Uma classe entidade modela informação e comportamento associado que geralmente tem uma vida longa. Este tipo de classe pode refletir uma entidade do mundo real, ou ela pode ser necessária para executar uma tarefa interna do sistema. Elas são tipicamente independentes do meio em que estão; isso é, elas não precisão saber como as classes que estão no limite se comunicam com o sistema. Muitas vezes, elas são aplicações independentes, isso significa que muitas delas poderão ser utilizadas por mais de uma sistema.

O primeiro passo é examinar as responsabilidades documentadas no fluxo de eventos para o caso de uso identificado (i.e., o que o sistema deve fazer). Classes entidade são normalmente utilizadas pelo sistema para definir alguma responsabilidade. Os nomes e as frases usadas para descrever a responsabilidade podem ser um bom ponto de partida. A lista inicial de nomes deve ser filtrada, pois ela pode conter nomes que estão fora do domínio do problema, nomes que são somente expressão de linguagem, nomes que são redundantes, e nomes que são descrições da estrutura da classe.

Classes entidade são normalmente encontradas na Fase de Elaboração. Elas são freqüentemente chamadas de classes de domínio, desde que elas usualmente tratam com abstrações de entidades do mundo real.

6.4. CLASSES LIMITE

Classes Limite cuidam da comunicação entre meio com o qual o sistema interage e o sistema propriamente dito. Elas fornecem a interface para um usuário ou para um outro sistema (i.e., a interface para um ator). Elas constituem a parte do sistema que dependem do meio em que elas estão. Classes limite são utilizadas para modelar as interfaces do sistema.

Cada par ator/cenário é examinado para descobrir as classes limite. As classes limites escolhidas na Fase de Elaboração do desenvolvimento são tipicamente de alto nível. Por

exemplo, você pode modelar uma janela mas não modela cada caixa de diálogo e botões. Neste ponto, você está documentando os requerimentos da interface com o usuário, não implementando a interface.

Os requerimentos das interfaces com o usuário tendem a ser muito vagos - os termos amigáveis e flexíveis são muito utilizados. Mas amigável, tem significados diferentes para pessoas diferentes. Neste caso as técnicas de prototipagem podem ser muito úteis. O cliente pode dar uma olhada e sentir o sistema e realmente entender o que o termo amigável significa. O "*o que*" é então compreendido assim como a estrutura e o comportamento da Classe Limite. Durante o projeto, essas classes são refinadas para levar em consideração os mecanismos da interface escolhida.

Classes Limite são também adicionadas para facilitar a comunicação com outros sistemas. Durante o projeto, estas classes são refinadas para levar em consideração o protocolo de comunicação escolhido.

6.5. CLASSES CONTROLE

Classes Controle modelam uma seqüência de comportamento específico a um ou mais casos de uso. Classes Controle coordenam os eventos necessários para a realização do comportamento especificado em um caso de uso. Você pode pensar que uma Classe Controle como "rodando" ou "executando" o caso de uso - ela representa a dinâmica do caso de uso. Estas classes normalmente são classes dependentes da aplicação.

Logo nos primeiros estágios da fase de Elaboração, uma Classe Controle é adicionada para cada par Ator/Caso de Uso. A Classe Controle é responsável pelo fluxo de eventos do caso de uso.

O uso de Classes Controle é muito subjetivo. Muitos autores sentem que o uso de Classes Controle resultam no comportamento sendo separado dos dados. Isso pode acontecer se a Classe Controle não foi escolhida prudentemente. Se uma Classe Controle está fazendo mais do estabelecer uma seqüência, então ela está fazendo coisas demais. Por exemplo, no sistema de matrículas, um estudante seleciona um curso ofertado, se o curso ofertado está disponível, o estudante é matriculado nele. Quem sabe como matricular um estudante - a Classe Controle ou a *OfertaCurso*? A resposta correta é, *OfertaCurso*. A Classe Controle sabe quando o estudante deveria ser matriculado, o curso ofertado sabe como matricular o estudante. Uma péssima Classe Controle não saberia só quando matricular o estudante mas como matricular o estudante.

A adição de uma Classe Controle para cada par Ator/Caso de Uso é somente um começo - enquanto a análise e o projeto continuam, as Classes Controle podem ser eliminadas, explodidas ou combinadas.

6.6. OBJETOS E CLASSES NO PROBLEMA DO SISTEMA DE MATRÍCULA (MATRI)

Vamos dar uma olhada no cenário *Adicionando uma Oferta de Curso para Lecionar*, o qual é um dos subfluxos do caso de uso *Selecionar Cursos para Ministrar* (ver apostila anterior, sobre Comportamento do Sistema). A principal definição especificada neste cenário é a habilidade do professor selecionar um curso ofertado para lecionar em um dado semestre.

Embora nós estejamos olhando este processo passo a passo, muitos desses passos podem ocorrer ao mesmo tempo no mundo real.

Identificando Classes Limite

Este caso de uso interage somente com o ator Professor. A ação especificada neste cenário é somente uma das definidas no caso de uso (o caso de uso também afirma que o Professor pode modificar, excluir, rever e imprimir a seleção). Isto significa que alguma coisa no sistema deve prover ao Professor a capacidade de selecionar a sua opção. Uma classe contém todas as opções disponíveis ao Professor como está registrado no caso de uso, para satisfazer a especificação feita. Esta classe é chamada *OpçõesCursoProfessor*. Adicionalmente podemos identificar uma classe que faça a adição de um novo Curso Ofertado para o Professor. Esta classe é chamada *AdicionaOfertaCurso*.

Identificando Classes Entidade

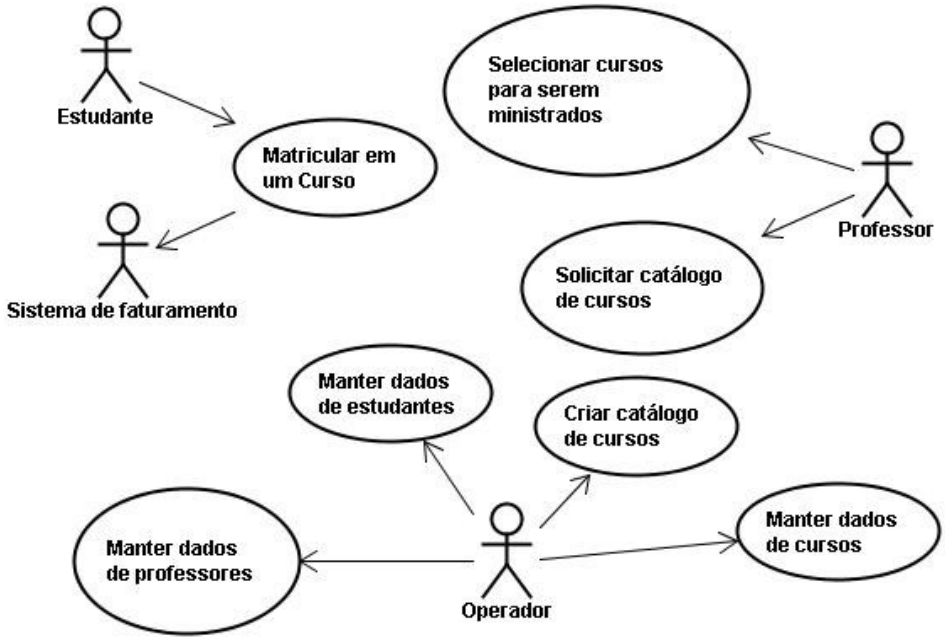
Este cenário está relacionado com Cursos, com as Ofertas de Curso e com o Relacionamento do Curso com o Professor. Nós podemos identificar três classes entidade: *Curso*, *OfertaCurso* e *InformaçãoProfessor*.

Identificando Classes de Controle

Iremos adicionar uma classe de controle para manusear o fluxo de eventos para o caso de uso. Esta classe é chamada *ControleCursoProfessor*. As classes identificadas foram adicionadas ao modelo.

6.7. ATIVIDADE

Baseando-se no modelo de Caso de Uso abaixo, tente levantar as classes para o sistema de Matrícula (MATRI).



AULA 7 IDENTIFICAÇÃO DAS RESPONSABILIDADES DAS CLASSES

7.1. APRESENTAÇÃO

Nesta aula será visto como identificar as classes de um sistema usando os cartões CRC

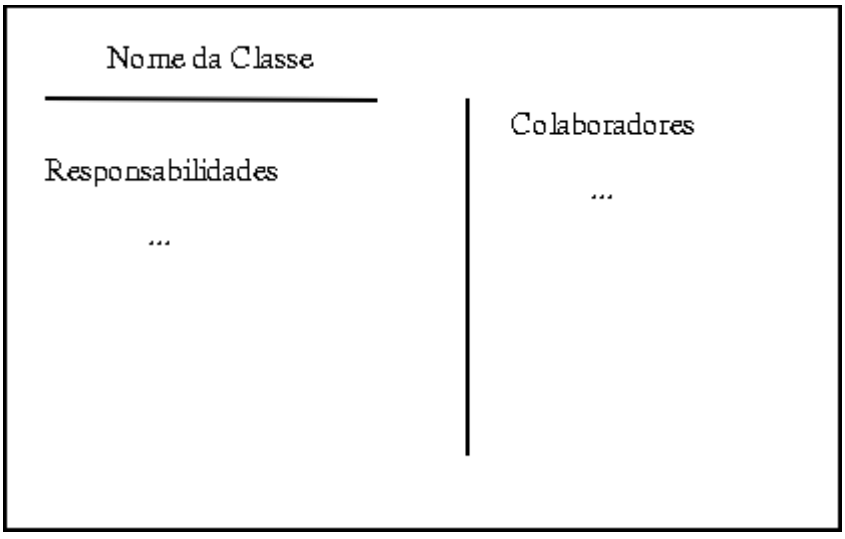
7.2. CARTÕES CRC

Uma forma para modelar um problema em classes e objetos é a forma proposta no artigo "A Laboratory For Teaching Object-Oriented Thinking", por ser simples e fácil de ser aplicada. São os Cartões Classe-Responsabilidade-Colaboradores (CRC). Os cartões CRC foram usados por algumas metodologias de desenvolvimento como a apresentada por Wirfs-Brock, Wilkerson e Wiener no livro "Designing Object-Oriented Software".

É baseada nos três atributos principais de um objeto na fase de projeto: nome da classe, suas responsabilidades e seus colaboradores.

- O nome da classe deve descrever o objeto no contexto geral do ambiente. Deve-se tomar um certo cuidado e gastar um pouco de tempo para escolher o conjunto certo de palavras para descrever o objeto.
- As responsabilidades devem identificar os problemas a serem resolvidos e não as soluções. Identificando-se os problemas fica fácil escolher entre as soluções possíveis. Novamente deve-se tomar cuidado com o conjunto de palavras usadas. As frases devem ser curtas e concisas.
- Os colaboradores de um objeto são os objetos para os quais este irá enviar mensagens a fim de satisfazer as suas responsabilidades.

Para facilitar a modelagem, Cunningham inventou os cartões Classe-Responsabilidades-Colaboradores (CRC). Esses cartões servem, também, como meio de documentação do projeto. Os cartões CRC são pedaços de papel grosso medindo 10x15cm e contém o nome da classe, as responsabilidades e os colaboradores, como na figura.



A disposição espacial dos cartões é importante. Quando duas classes são mútuas colaboradoras, seus cartões devem ser colocados levemente sobrepostos. Quando uma classe é colaboradora de outras classes (a classe é usada pelas outras), seu cartão deve ficar abaixo dos cartões das outras classes. Se há uma relação de herança, deverá haver uma sobreposição quase completa dos cartões. A disposição dos cartões é importante, pois em projetos ainda incompletos é possível notar a localização de uma classe antes da mesma ter sido projetada.

Para achar as classes, sua responsabilidades e seus colaboradores deve-se seguir as seguintes etapas:

1. ·Definição das classes e das estruturas de dados de cada classe;
2. ·Definição das responsabilidades de cada classe;
3. ·Definição dos colaboradores de cada classe;

O que são classes e como defini-las

Num problema a ser resolvido, as diversas classes podem ser identificadas como as entidades que existem no problema, por exemplo, aluno, turma, professor, etc... São classes, também, as estruturas de dados utilizadas para resolver o problema, bem como os dispositivos(físicos e virtuais) a serem acessados. Por exemplo: interface, arquivo, controlador de tempo, etc...

O que são responsabilidades e como defini-las

As responsabilidades são definidas como sendo os problemas que as classes devem resolver. São, a grosso modo, as funções das classes. Por exemplo: colocar um registro no arquivo, dar presença ao aluno, cobrar mensalidade, etc... As responsabilidades são os problemas a serem resolvidos. A maneira de resolve-los vão ser os métodos das classes(implementação).

O que são colaboradores e como defini-los

Os colaboradores são as classes que ajudam uma classe a resolver as suas responsabilidades. São as classes que vão prestar serviços à classe. A classe que está sendo analisada solicita serviços a outras classes e estas, por sua vez, prestam estes serviços, ajudando a resolver os problemas.

7.3. ATIVIDADE

1. modelar o problema de controle de uma pizzaria de entrega em domicílio. O cliente pede uma ou mais pizzas, refrigerantes ou cervejas. O atendente anota o pedido no sistema. Cada pizza pode ser (média, grande ou enorme). Cada pizza pode ter no máximo 3 sabores. O atendente anota os dados do cilente e a forma de pagamento (inclusive o troco necessário). O atendente anota o código do entregador e o tempo de entrega. O gerente cadastra os sabores de pizzas existentes, os dados dos entregadores e visualizaos relatórios de vendas(por sabor, região) e tempo de entrega.

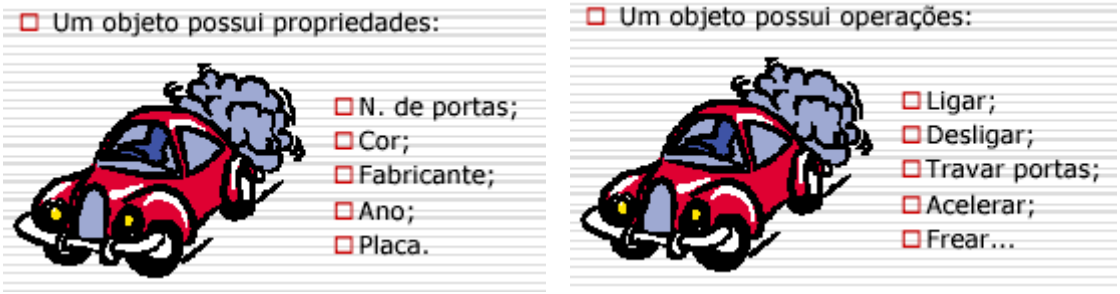
AULA 8 IDENTIFICAÇÃO DE ATRIBUTOS E OPERAÇÕES DE UMA CLASSE

8.1. APRESENTAÇÃO

Nesta aula será visto como identificar os atributos e as operações das classes do sistema

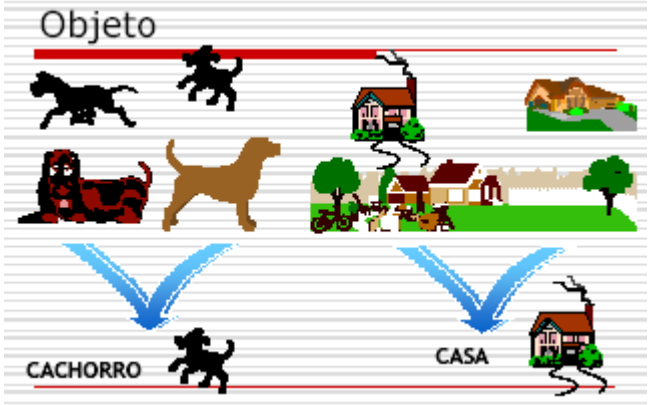
8.2. ORIENTAÇÃO A OBJETOS (OO)

É uma forma de entender e representar sistemas complexos como estruturas hierárquicas de objetos correlatos.



Objeto

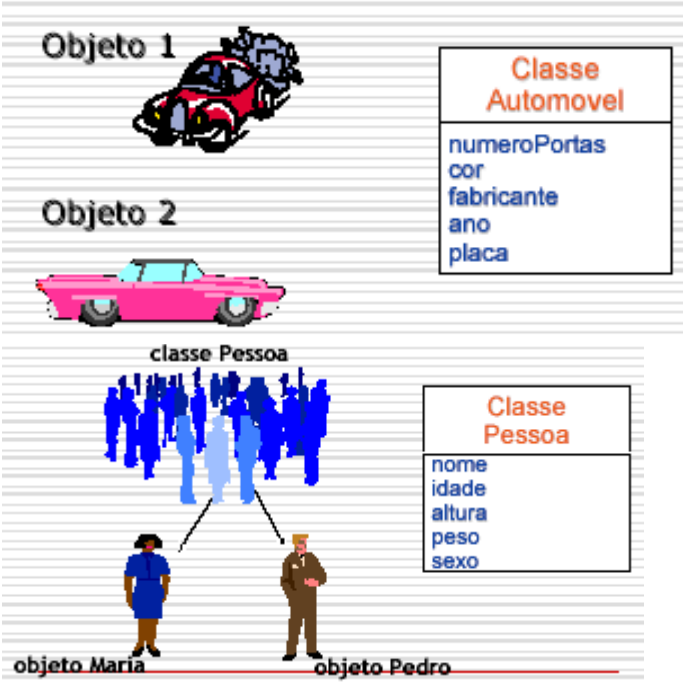
- Um objeto é uma construção de software que encapsula estado e comportamento, através de propriedades (atributos) e operações (métodos);
- Estado de um objeto: composto por suas propriedade e seus respectivos valores;
- Comportamento: a maneira como o objeto reage quando o seu estado é alterado ou quando uma mensagem é recebida.



CLASSES

- Conjunto de objetos similares.
    - Estrutura de dados similares (propriedades);

    - Comportamento similar (operações)
- Um objeto é uma instância de uma classe;
- Objetos de uma mesma classe diferenciam-se pelos valores de suas propriedades e de seus identificadores únicos.

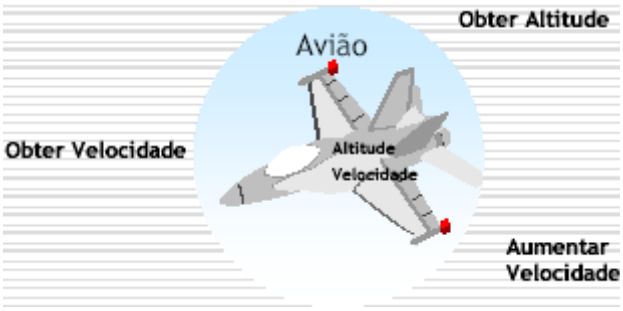


Troca de Mensagens

- Mecanismo através do qual os objetos se comunicam, invocando as operações desejadas;
- Um objeto (Emissor) envia uma mensagem a outro (Receptor) que executará uma tarefa;
- Operações (métodos) são invocados através de mensagens.

Encapsulamento

- “Não mostre as cartas de seu baralho”
- Objetivo: Ocultar do mundo externo ao objeto, os detalhes de implementação e restringir o acesso aos propriedades e métodos.
- Vantagens:
    - Segurança no acesso ao objeto;
    - Melhor consistência no estado interno, pois evita alterações incorretas de valores das propriedades.



8.3. O QUE É UMA OPERAÇÃO?

- Uma classe incorpora um conjunto de responsabilidades que definem o comportamento dos objetos na classe
- As responsabilidades de uma classe são executadas por suas operações
  - Não é necessariamente um mapeamento um-para-um
    - Responsabilidade da classe Produto -- fornecer preço
    - Operações para esta responsabilidade
      - Buscar informação de um banco de dados
      - Calcular o preço
- Uma operação é um serviço que pode ser requisitado por um objeto para obter um dado comportamento

Operações Dependem do Domínio

- Liste todas as operações relevantes ao domínio do problema
  - As operações de uma classe Pessoa serão diferentes dependendo de 'quem está perguntando'

Perspectiva do Bancário	Perspectiva do Médico
receber empréstimo	examinar
anexar conta	tomarRemédio
receber linhaDeCrédito	irParaHospital
	receberConta

Nomeando Operações

- As operações devem ser nomeadas para indicar seus resultados, não os passos por trás da operação. Exemplos:
  - calcularSaldo()
    - Nome pobre
      - Indica que o saldo deve ser calculado - isto é uma decisão de implementação/otimização
  - obterSaldo()
    - Bem nomeado
      - Indica apenas o resultado

Nomeando Operações

- As operações devem ser nomeadas do ponto de vista do fornecedor e não do cliente
- Em um posto de gasolina, a gasolina vem da bomba
  - Uma operação para que a bomba tenha esta responsabilidade -- como

deveria ser chamada?
- Bons nomes -- distribuir(), darGasolina()
- Nome ruim -- receberGasolina()
  - A bomba dá a gasolina -- ela não recebe a gasolina

O que é uma Operação Primitiva?

- Uma operação primitiva é uma operação que pode ser implementada apenas usando o que é intrínseco, interno a classe
  - Todas as operações de uma classe são tipicamente primitivas
- Exemplos:
  - Adicione um item a um conjunto -- operação primitiva
  - Adicione quatro itens a um conjunto -- não primitiva

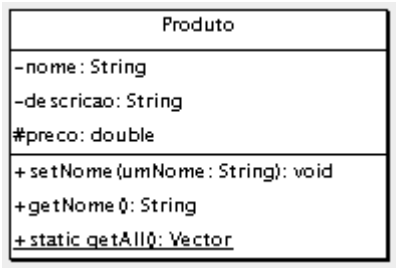
Pode ser implementada com múltiplas chamadas a operação adicione um item a um conjunto

Assinatura da Operação

- A assinatura da operação consiste de :
  - Lista de argumentos opcional
  - Classe de retorno
- Durante a análise NÃO É OBRIGATÓRIO preencher a assinatura de operações
  - Esta informação pode ser adiada para fase de projeto

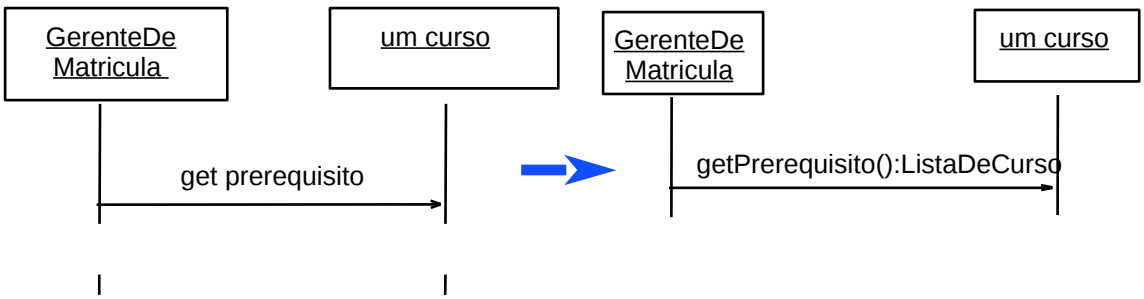
Mostrando Operações

▪Operações são mostradas no terceiro compartimento da classe



Descobrindo Operações nos Diagramas de Interação

- As mensagens mostradas nos diagramas de seqüência e/ou colaboração são usualmente operações da classe receptora
- As mensagens são traduzidas em operações e adicionadas ao diagrama de classes



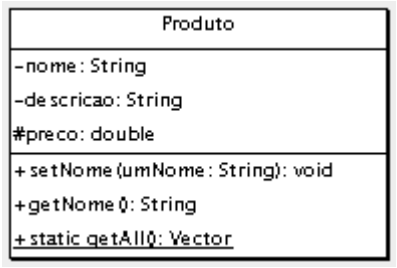
8.4. O QUE É UM ATRIBUTO?

- Um atributo é uma característica de uma classe

- Atributos não tem comportamento -- eles não são objetos
- Nomes de atributos são substantivos simples ou frases substantivas
  - Os nomes devem ser únicos dentro da classe
- Cada atributo deve ter uma definição clara e concisa
- Bons atributos para a classe Estudante:
  - Nome -- primeiro e último nome
  - CampoEstudo -- principal campo de estudo
- Atributos ruins -- cursosSelecionados
  - Isto é um relacionamento e não um atributo

Mostrando Atributos

- Atributos são mostrados no segundo compartimento da classe



Como Descobrir Atributos?

- Muitos atributos são descobertos no texto da descrição do fluxo de eventos para os casos de uso
  - Procure substantivos que não foram considerados bons candidatos para classes
- Outros são descobertos quando a definição da classe é criada
- Experiência no domínio também pode prover bons atributos

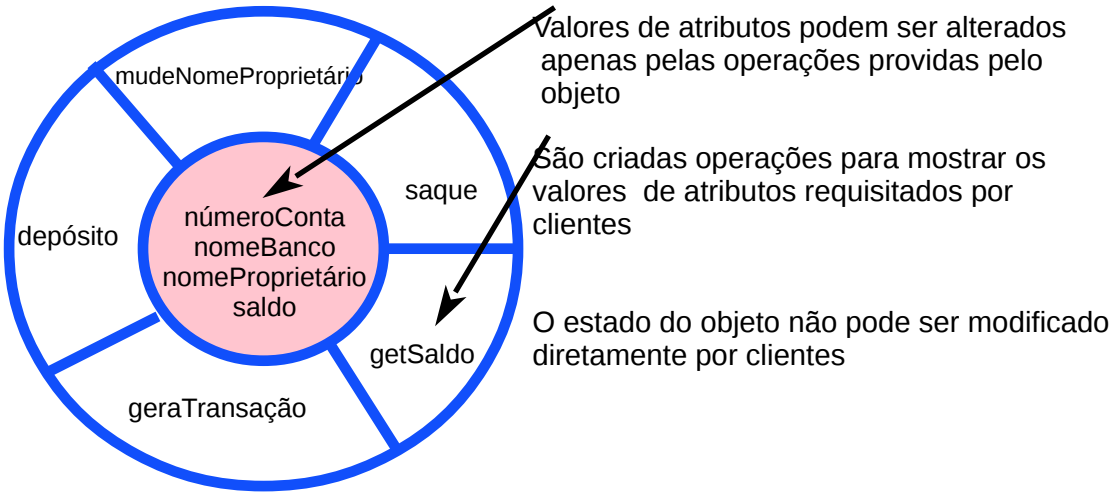
Exibindo Atributos e Operações

- Atributos e/ou operações podem ser mostrados dentro de uma classe
- Diagramas de classes adicionais podem ser criados para exibir atributos e operações
  - Os relacionamentos não costumam ser exibidos nestes diagramas de classes

8.5. ENCAPSULAMENTO

- Uma maneira de visualizar uma classe é a que consiste de duas partes: a interface e a implementação
  - A interface pode ser vista e usada por outros objetos (clientes)
  - A implementação é escondida dos clientes
- Esconder os detalhes da implementação de um objeto é chamado encapsulamento ou "information hiding"
- Encapsulamento oferece dois tipos de proteção. Protege:
  - O estado interno de um objeto de ser corrompido por seus clientes
  - O código cliente de mudanças na implementação dos objetos

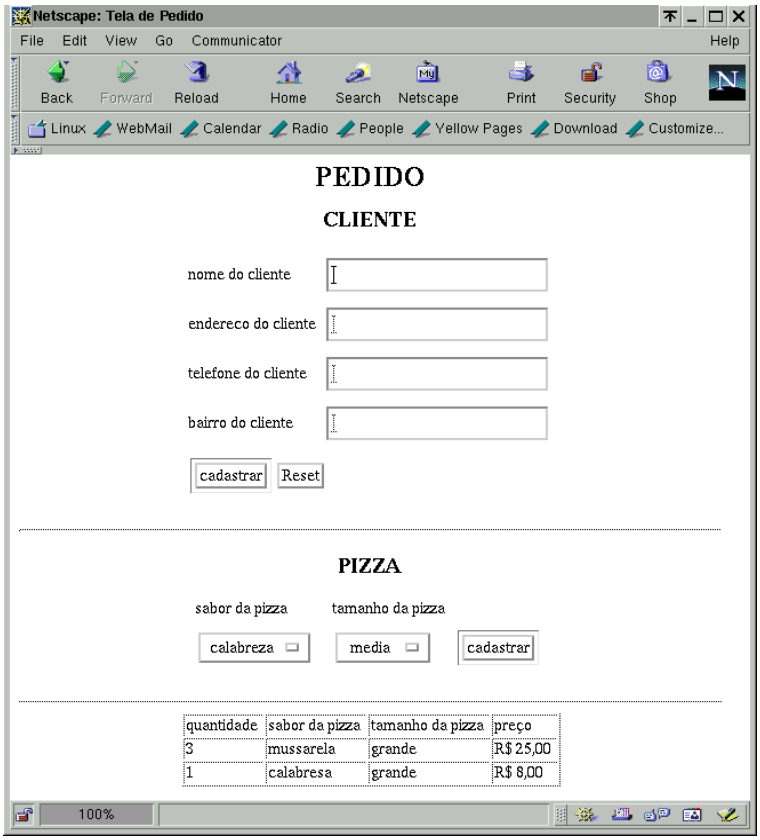
Exemplo: Encapsulamento



Benefícios do Encapsulamento

- O código cliente pode usar a interface para uma operação
- O código cliente não pode tirar vantagem da implementação de uma operação
- A implementação pode mudar, por exemplo, para:
  - Corrigir um erro (*bug*)
  - Aumentar performance
  - Refletir uma mudança no plano de ação
- O código cliente não será afetado pelas mudanças na implementação, assim reduzindo o "efeito ondulação" (ripple effect) no qual uma correção em uma operação força correções correspondentes numa operação cliente, o qual por sua vez, causa mudanças em um cliente do cliente...
- A manutenção é mais fácil e menos custosa

8.6. ATIVIDADE



1) modelar o problema de controle de uma pizzaria de entrega em domicílio. O cliente pede uma ou mais pizzas. O atendente anota o pedido no sistema. Cada pizza pode ser (média, grande ou enorme). Cada pizza tem apenas 1 (um sabor). O atendente anota os dados do cilente e o pedido. A tela acima é a usada para anotar o pedido.

AULA 9 ASSOCIAÇÕES ENTRE CLASSES

9.1. APRESENTAÇÃO

Nesta aula iremos abordar e exercitar o que são associações entre classes e como usá-las. Iremos abordar aspectos como: associações, nomes, papeis, multiplicidade, qualificadores, navegabilidade e associações reflexivas.

9.2. ASSOCIAÇÕES

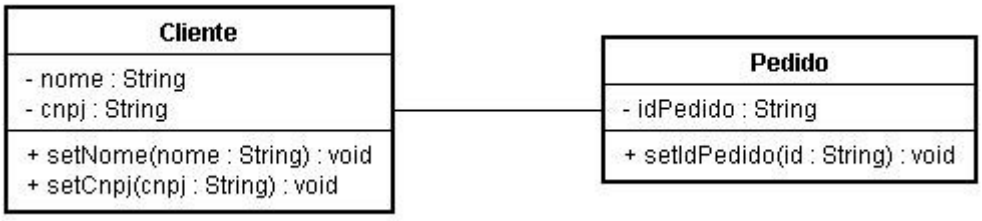
A Necessidade de Relacionamentos
Todos os sistemas abrangem muitas classes e objetos
Objetos atuam no comportamento de um sistema colaborando entre eles

- A Colaboração é realizada através de relacionamentos

Ocorrem dois tipos importantes de relacionamentos durante a análise

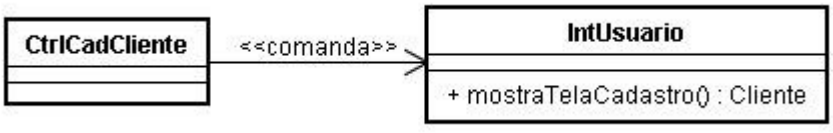
- Associação e
- Generalização

Associações
Uma associação é uma conexão semântica bi-direcional entre classes
Isto implica na existência de uma ligação (link) entre os objetos das classes associadas
Associações são representadas no diagrama de classes por uma linha ligando as classes associadas. Em um link os dados podem fluir em ambas as direções

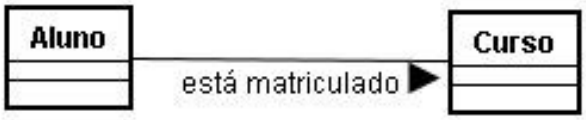


Navegação
Uma associação é um relacionamento bi-direcional

- Dada uma instância de GerentelDeMatrícula há um objeto Curso associado
- Dada uma instância de Curso há um objeto OficialDeMatrícula associado



Dando Nomes às Associações
Para tornar seu significado mais claro, uma associação pode receber um nome
O nome é representado como uma etiqueta (label) colocada ao longo da linha de associação, no meio da linha, entre os ícones das classes
Um nome de associação geralmente é um verbo ou uma frase verbal

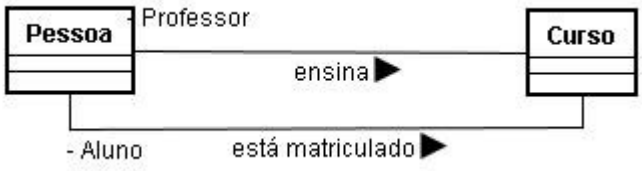


Definindo Papéis
Um papel denota o propósito ou capacidade em que uma classe se associa com outra

- Nomes de papéis são tipicamente substantivos ou frases substantivas
- Um nome de papel é colocado ao longo da linha de associação, próximo da classe referenciada
- Um ou ambos os lados da associação podem ter nomes de papéis

Associações Múltiplas

- Pode existir mais de uma associação entre duas classes
- Se houver mais de uma associação entre duas classes, elas DEVEM ser nomeadas



Associações múltiplas devem ser questionadas

9.3. MULTIPLICIDADE PARA ASSOCIAÇÕES

Multiplicidade é o número de instâncias de uma classe relacionada a UMA instância da outra classe
Para cada associação, devem ser tomadas duas decisões de multiplicidade : uma para cada lado da associação

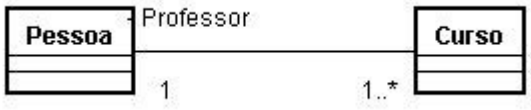
Por exemplo, na conexão entre Pessoa, atuando no papel de professor, e Curso

- Para cada instância de Pessoa, muitos (i.e., zero ou mais) Cursos podem ser ministrados
- Para cada instância de Curso, exatamente uma Pessoa é o professor

Muitos	*
Exatamente Um	1
Zero or mais	0..*
Um ou Mais	1..*
Zero or um	0..1
Intervalo Específico	2..4

Indicadores de Multiplicidade
Cada lado de uma associação contem um indicador de multiplicidade
Indica o número de objetos que participam no relacionamento

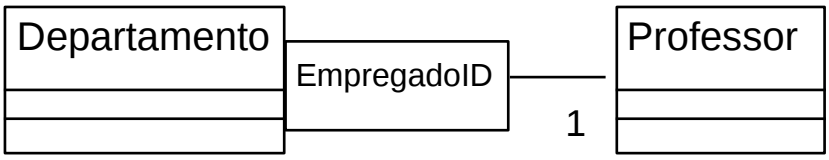
Exemplo: Multiplicidade



Decisões de Multiplicidade expõem muitas suposições, antes ocultas, sobre o problema sendo modelado

- Um professor pode estar indisponível?
- Um curso pode ter dois professores?

Qualificadores
Um qualificador é um atributo de uma classe que pode ser usado para reduzir a multiplicidade da associação



Restrições

- Uma restrição expressa alguma condição que deve ser preservada
  - Uma restrição é mostrada entre chaves.

9.4. ATIVIDADE

1. Faça um Diagrama de Classes do subsistema de Operação de Trem, sem utilizar herança e considerando o seguinte:

   - Um trem pode ser de carga, passageiros ou ambos e pode ser movido por um ou mais vagões com força motriz.
   - Um vagão com força motriz possui um determinado tipo de motor, com potência específica.
   - O trem de passageiros pode possuir várias portas por vagão, que devem abrir e fechar automaticamente, ao chegar na estação e antes de partir, respectivamente.
   - A capacidade de passageiros de um trem deve estar registrada em quantidade, e a capacidade de cada vagão de carga em metros cúbicos.

- Cada trem possui uma rota que deve ser seguida durante a viagem.
- A data de início de trabalho de cada vagão deve ser registrada.
- A velocidade do trem deve ser controlada automaticamente. Ao arrancar ele deverá estar ganhando velocidade, e ao logo da viagem deverá manter outra velocidade e estando a uma determinada distância da estação deverá reduzir a velocidade para parar.
- Os segmentos de trilho iniciam e terminam em um determinado quilômetro.
- A localização do trem numa ferrovia deve ser possível.

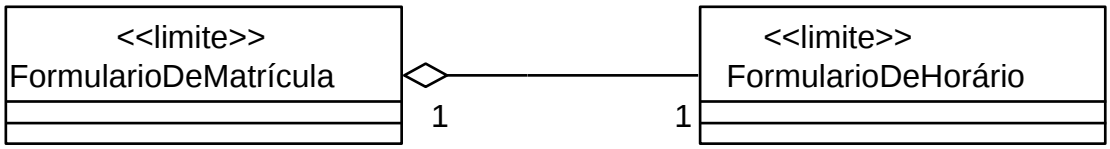
AULA 10 AGREGAÇÕES E ASSOCIAÇÕES

10.1. APRESENTAÇÃO

Nesta aula iremos abordar e exercitar o que são agregações entre classes e como usá-las. Iremos abordar aspectos como: agregações, composições, associações reflexivas, classes de ligação e as diferenças entre associações e agregações.

10.2. AGREGAÇÃO

Agregação é uma forma especializada de associação na qual o todo está relacionado a sua(s) parte(s). Agregação é conhecida como “parte-de” ou relacionamento de conteúdo. Uma agregação é representada como uma associação com um losango perto da classe que denota a agregação (todo). A multiplicidade é representada da mesma maneira que nas outras associações

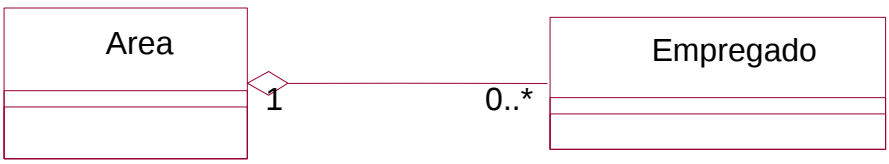


Testes de Agregação

- A frase “parte de” é usada para descrever o relacionamento?
  - Uma Porta é “parte de” um Carro
- Algumas operações no todo são automaticamente aplicadas nas partes?
  - Mova o Carro, Mova a Porta
- Alguns valores de atributos são propagados do todo para algumas ou todas as suas partes?
  - O Carro é azul, a Porta é azul
- Há uma assimetria intrínseca no relacionamento onde uma classe é subordinada a outra?
  - Uma Porta É parte de um Carro, um Carro NÃO É parte de uma Porta

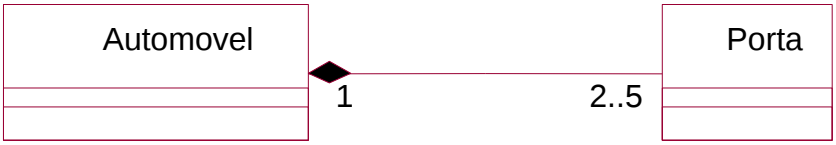
Tipos de Agregação

- Existem dois tipos de agregação:
  - Agregação propriamente dita, ou também chamada “agregação por referência”
    - Conhecida como relacionamento “tem-um(a)”
    - Envolve partes componentes que existem independentemente de seus agregados
    - Exemplo: A área “X” da empresa tem os empregados “A”, “B” e “C”



Visualizando o conceito de agregação, o agregado contém uma referência aos seus componentes

- Composição, ou também chamada “agregação por valor”
  - Conhecida como relacionamento “contém um(a)”
  - Especifica que as partes componentes só tem um dono
  - Especifica que o composto possui suas partes componentes
  - Especifica que as partes componentes existem, ou “vivem e morrem” com seu composto proprietário
  - Exemplo: Um automóvel possui de 2 a 5 portas



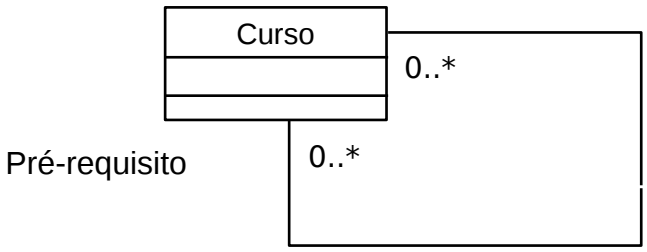
- Visualizando o conceito de composição, o composto contém os seus componentes

Associação ou Agregação ?

- Se dois objetos são estreitamente ligados por um relacionamento parte-todo
  - O relacionamento é uma agregação
- Se dois objetos são usualmente considerados independentes, mesmo que eles estejam freqüentemente ligados
  - O relacionamento é uma associação

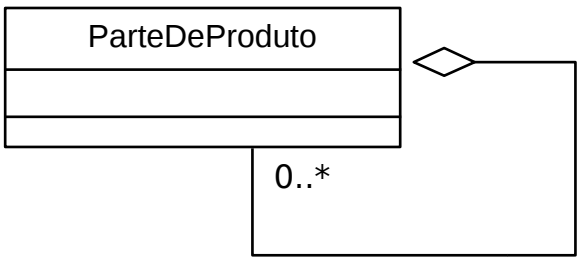
10.3. ASSOCIAÇÕES REFLEXIVAS

- Em uma associação reflexiva, objetos de uma mesma classe são relacionados
  - Indica que múltiplos objetos da mesma classe colaboram juntos de algum modo



- Um curso pode ter muitos pré-requisitos
- Um curso pode ser um pré-requisito para muitos outros cursos

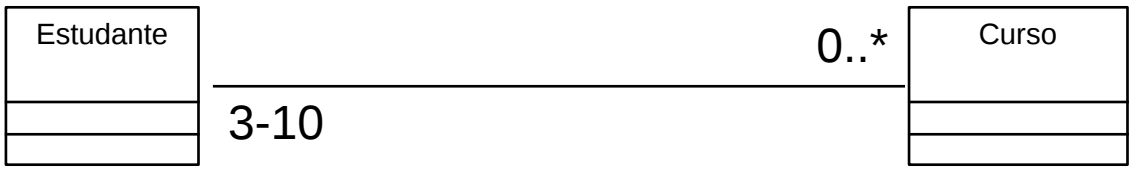
- Agregados também podem ser reflexivos
  - Problema clássico de lista de materiais
- Isto indica um relacionamento recursivo



Um objeto ParteDeProduto é “composto de” zero ou mais objetos ParteDeProduto

10.4. CLASSES DE LIGAÇÃO

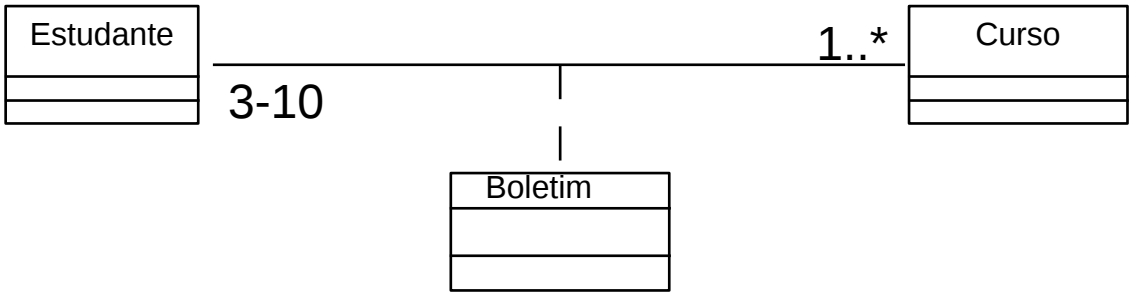
Nós desejamos rastrear todas as notas que um estudante teve em todos os cursos que fez
O relacionamento entre Estudante e Curso é um relacionamento muitos-para-muitos
Onde colocamos o atributo nota?



O atributo nota não pode ser colocado em Curso porque há (potencialmente) muitas ligações para muitos objetos Estudante
O atributo nota não pode ser colocado na classe Estudante porque há (potencialmente) muitas ligações para muitos objetos Curso
Portanto, o atributo realmente pertence a uma ligação particular Estudante-Curso
Uma classe de ligação é usada para conter a informação da ligação

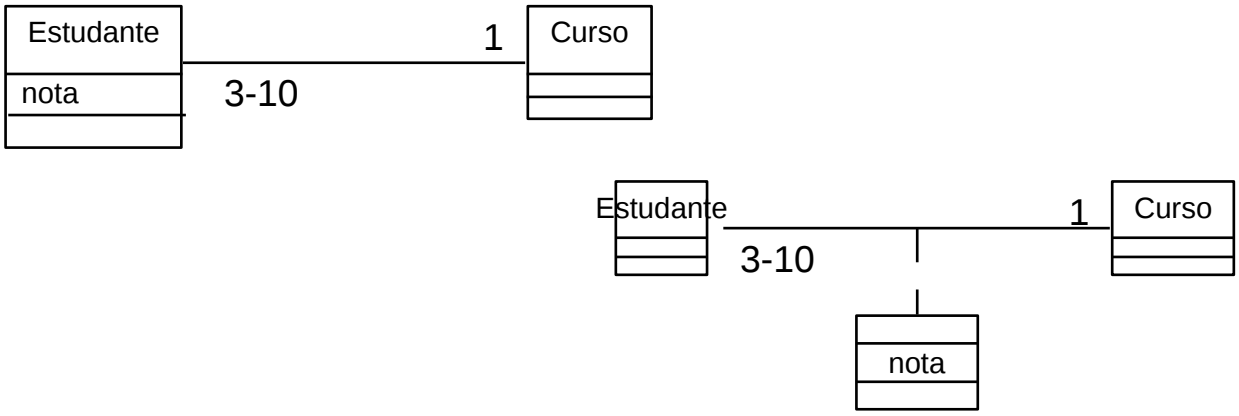
Desenhando Classes de Ligação

Crie uma classe de ligação usando um ícone de classe
Conecte a classe na linha da associação usando uma linha tracejada
A classe de ligação pode incluir propriedades múltiplas da associação
Apenas uma classe de ligação é permitida por associação



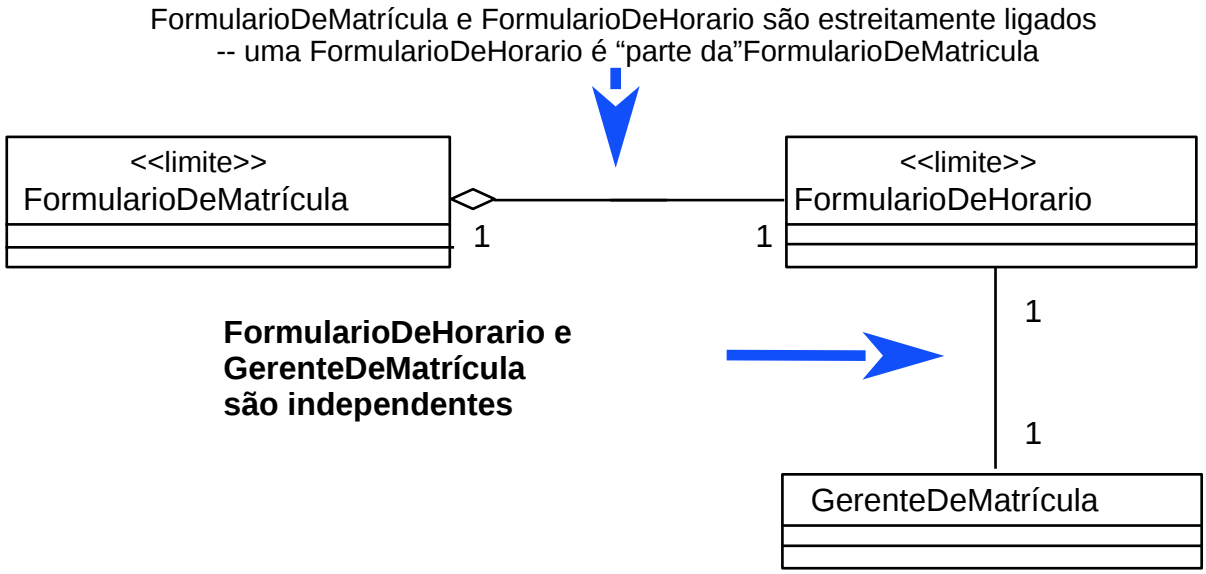
Classes de Ligação e Multiplicidade

Classes de Ligação são freqüentemente usadas em associações muitos-para-muitos
Se a multiplicidade em um os lados de uma associação é “para-um”
O atributo pode ser colocado dentro da classe no lado muitos do relacionamento
OU
Uma classe de ligação ainda pode ser usada



10.5. ENCONTRANDO ASSOCIAÇÕES E AGREGAÇÕES

Associação ou Agregação ?



10.6. ATIVIDADE

Faça um Diagrama de Classes para um problema com as seguintes características:

- Existem medicamentos aprovados para uso clínico, pelo órgão governamental apropriado.
- Esses medicamentos são fabricados por firmas capacitadas para tanto, que atribuem aos medicamentos um nome de medicamento do fabricante, além do nome genérico que os mesmos já possuem.
- Nem todos os medicamentos podem ser fabricados por todas as empresas.
- A data em que uma empresa foi outorgada para fabricar um medicamento indica o início da permissão para fabricação.
- Cada fabricante pode produzir muitos lotes do medicamento, que tem data de fabricação e validade.
- O modelo deve estar apto a responder por quem um medicamento (ex.: ácido acetil-salicílico) foi fabricado, em que lote(s) e com que número de permissão.

AULA 11 HERANÇA

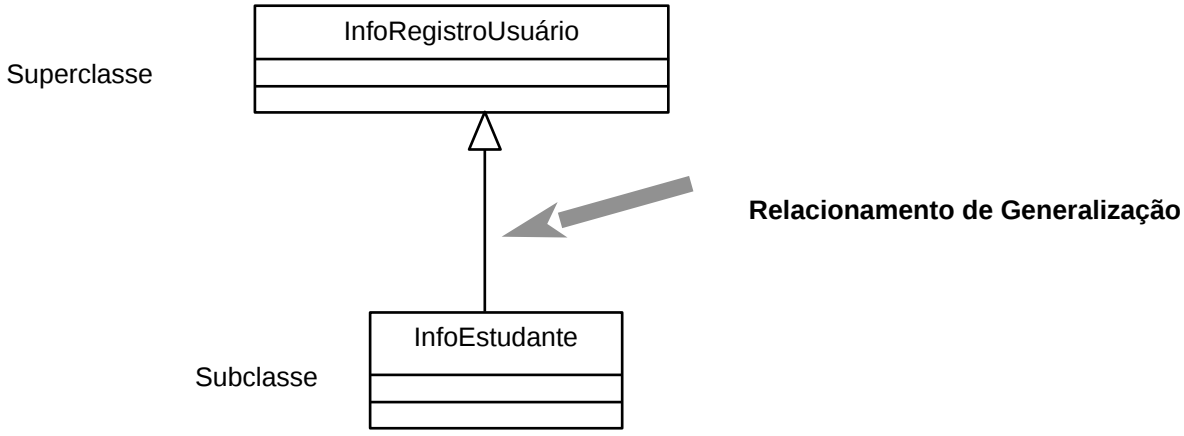
11.1. APRESENTAÇÃO

Nesta aula iremos abordar e exercitar o que são generalizações entre classes e como usá-las. Iremos abordar aspectos como: generalização, hierarquia, notação, subclasse, superclasse, especialização, e herança múltipla.

11.2. GENERALIZAÇÃO

- **Generalização** define um relacionamento entre classes onde uma classe compartilha a estrutura e/ou comportamento de uma ou mais classes
- Generalização define uma hierarquia de abstrações na qual uma subclasse herda de uma ou mais superclasses
  - Com **herança simples**, a subclasse herda de apenas uma superclasse
  - Com **herança múltipla**, a subclasse herda de mais de uma superclasse
- Generalização é um relacionamento "é um" ou "tipo de"

Desenhando uma Hierarquia de Herança



Considerações sobre generalização

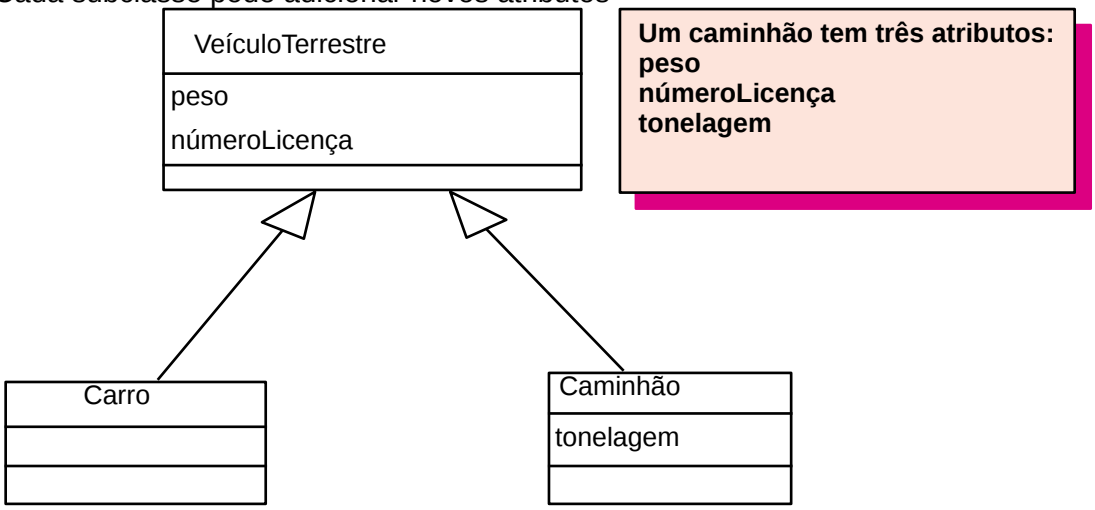
- Como um relacionamento de generalização não se refere a objetos individuais
  - O relacionamento não é nomeado
  - Multiplicidade não tem sentido
- Teoricamente, não há limite no número de níveis em uma hierarquia
- Na prática, os níveis precisam ser bem limitados
  - Hierarquias típicas em C++ tem 3 a 5 níveis
  - Hierarquias em Smalltalk podem ser um pouco maiores

O que é Herdado?

- Uma subclasse herda de seus pais:
  - Atributos
  - Operações
  - Relacionamentos
- Uma subclasse pode:
  - Incluir atributos, operações e relacionamentos adicionais
  - Redefinir as operações herdadas (use com cautela!)

Herdando Atributos

- Atributos são definidos no nível mais alto da hierarquia de herança na qual eles são aplicáveis
- Subclasses de uma classe herdam todos os atributos
- Cada subclasse pode adicionar novos atributos



Herdando Operações

- Operações são definidas no nível mais alto da hierarquia de herança na qual elas são aplicáveis
- Subclasses de uma classe herdam todas as operações
- Cada subclasse pode aumentar ou redefinir operações herdadas

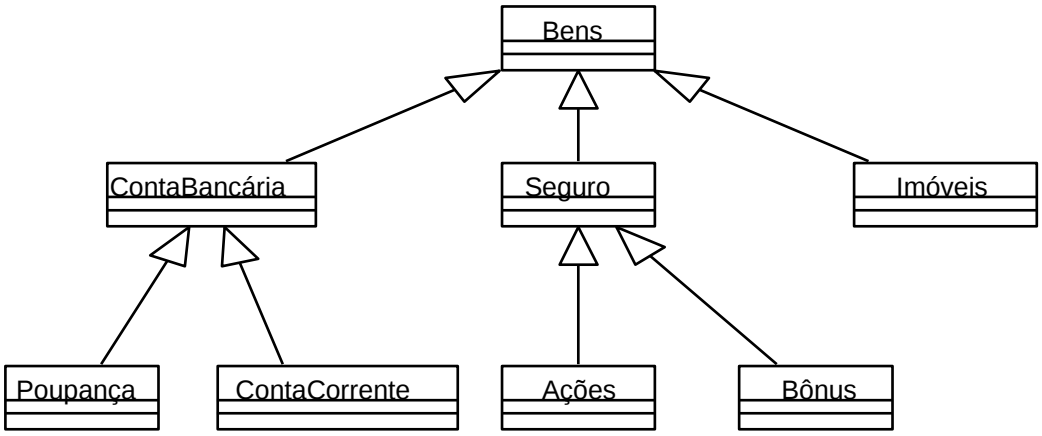
Herdando Relacionamentos

- Relacionamentos também são herdados e devem ser definidos no nível mais alto da hierarquia de herança na qual eles são aplicáveis
- Subclasses de uma classe herdam todos os relacionamentos
- Cada subclasse pode também possuir relacionamentos adicionais

Generalização de Classes

- Generalização proporciona a capacidade de criar superclasses que reunem estrutura e/ou comportamento comum a várias subclasses
- Procedimento de generalização
  - Identificar similaridades de estrutura/comportamento entre várias classes
  - Criar uma superclasse para reunir a estrutura/comportamento comum
- As classes originais passam a ser subclasses da nova superclasse
- Superclasses são mais abstratas que suas subclasses

Exemplo de Generalização



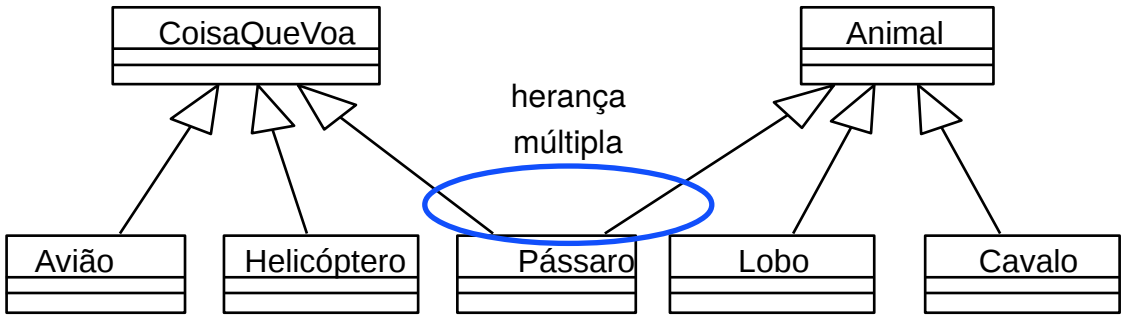
Especialização de Classes

- A especialização proporciona a capacidade de criar subclasses que representam refinamentos nos quais a estrutura e/ou comportamento da superclasse são adicionados ou modificados
- Procedimento de Especialização
- Notar que algumas instâncias apresentam estrutura ou comportamento especializado
- Criar subclasses para agrupar instâncias de acordo com sua especialização
- Subclasses são menos abstratas que suas superclasses

Hierarquias de Herança

- Tanto generalização quanto especialização são usadas no desenvolvimento de uma hierarquia de herança
- Durante a análise, são estabelecidas hierarquias de herança entre abstrações chaves (i.e., classes) se necessário
- Durante o projeto, as hierarquias de herança são refinadas para:
- Aumentar reutilização
- Incorporar classes de implementação
- Incorporar bibliotecas de classes disponíveis

11.3. HERANÇA MÚLTIPLA



Conceitos de Herança Múltipla

- Conceitualmente é necessário para modelar o mundo real de forma precisa
- Na prática, isto pode gerar dificuldades na implementação
- Nem todas as linguagens de programação orientadas a objetos suportam herança múltipla diretamente

Cada ambiente/linguagem de programação escolhe maneiras de resolver estas dificuldades

Encontrando generalização

- É importante avaliar todas as classes para encontrar possíveis generalizações
  - Procure por comportamento comum (operações) e estado comum (atributos) nas classes
- Técnica de adição
  - Adicione novas operações/atributos na(s) subclasse(s)
- Técnica de modificação
  - Redefina operações
  - Deve ser feito com cuidado para não alterar a semântica

Generalização versus Agregação

- Generalização e agregação são geralmente confundidas
  - Generalização representa um relacionamento "é-um" ou "tipo-de"
  - Agregação representa um relacionamento "tem-um"

Generalização	Agregação
Palavra chave **"é um"**	Palavra chave **"tem um"**
Relaciona objetos da mesma superclasse	Relaciona objetos de classes diferentes
Representado por uma seta	Representado por um losango

11.4. ATIVIDADE

Construa um Diagrama de Classes inicial para controlar empréstimos e reservas de publicações em uma biblioteca, considerando o seguinte:

- A biblioteca pertence a uma empresa e somente os funcionários da empresa estão aptos a reservar ou emprestar publicações.
- A biblioteca tem acesso aos dados cadastrais dos funcionários.
- Uma publicação pode ser um livro, uma revista, um jornal, etc. A biblioteca pode possuir vários exemplares de uma mesma publicação.
- Qualquer tipo de publicação pode ser emprestada ou reservada.
- Não há número limite de reservas por publicação.
- Quando uma publicação for devolvida e para ela existirem reservas, o primeiro da fila deve ser avisado.
- A reserva expira um dia após o aviso da disponibilidade da publicação ao interessado. Caso o mesmo não venha a emprestar a publicação, ela será considerada disponível e o próximo da fila deve ser avisado.
- Uma reserva pode ser excluída a pedido do usuário.
- Empréstimos de publicações não devolvidas 2 dias após o prazo serão consideradas irregulares e um funcionário da biblioteca deve resgatar o livro.

AULA 12 INTERFACES E PACOTES

12.1. APRESENTAÇÃO

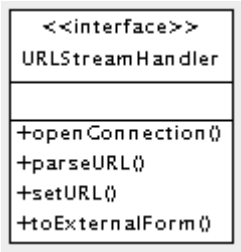
Nesta aula iremos abordar e exercitar o que são Intefaces e pacotes.

12.2. INTERFACES

Uma Interface é uma coleção de operações utilizadas para especificar um serviço de uma classe ou componente.

As interfaces são empregadas para visualizar, especificar, construir e documentar a coesão interna do sistema.
Escolhendo as interfaces corretas, você pode selecionar componentes padrão, bibliotecas e *frameworks* para a implementação destas interfaces, sem precisar construí-las. À medida que descobrir implementações melhores, você pode substituir as antigas sem perturbar seus usuários
Declarando a interface, você pode estabelecer o comportamento desejado de uma abstração independente de qualquer implementação desta interface.

Notação:



Relacionamentos:

A relação entre Alvo e Observador é uma relação de dependência, enquanto que a relação entre Observador e Rastreador é uma relação de realização. Uma relação de realização indica que a classe Rastreador implementa a interface Observador.

12.3. PACOTES

Se um sistema contém somente poucas classes, você pode gerenciá-las facilmente. Mas, a maioria dos sistemas são compostos por muitas classes, neste caso então é necessário um mecanismo para agrupa-las, para obter uma melhor facilidade de uso, manutenção e reutilização. Neste caso é onde o conceito de packages é útil. Um package na visão lógica de um modelo é uma coleção de packages e/ou classes relacionados. Agrupando classes em packages, nós podemos ter uma visão de mais alto nível do modelo (i.e., os packages), ou podemos nos aprofundar no modelo, dando uma olhada no que está contido no package.

Cada package contém uma interface que é implementada por um conjunto de classes públicas - aquelas classes com as quais os outros packages falam. O resto das classes em um package são classes de implementação - classes, que não se comunicam com as classes de outros packages.

Se um sistema é complexo, packages podem ser criados logo no início da Fase de Elaboração para facilitar a comunicação. Para sistemas simples, as classes descobertas na análise podem ser agrupadas em um package - o próprio sistema. No decorrer do processo de análise e projeto, o conceito de package será utilizado para agrupar as classes que são necessárias para realizar as decisões arquiteturais tomadas para o sistema.

Na UML, packages são representados como pastas.

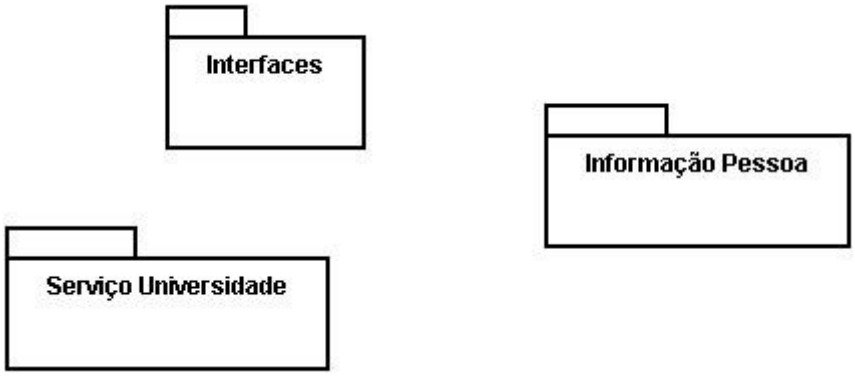
Criando Packages

O próximo passo é agrupar as classes em packages. Neste ponto, nós temos identificadas seis classes: *Curso*, *OfertaCurso*, *InformaçãoProfessor*, *OpçõesCursoProfessor*, *AdicionaOfertaCurso* e *ControleCursoProfessor*. Elas caem em três grupos lógicos - coisas únicas para a universidade, coisas que contém informações sobre pessoas e coisas que são parte das interfaces com os atores. Nós podemos identificar os packages: *Interfaces*, *ServicoUniversidade* e *InformacaoPessoa*. As classes são realocadas para os packages identificados. Os packages com suas classes são mostrados abaixo.

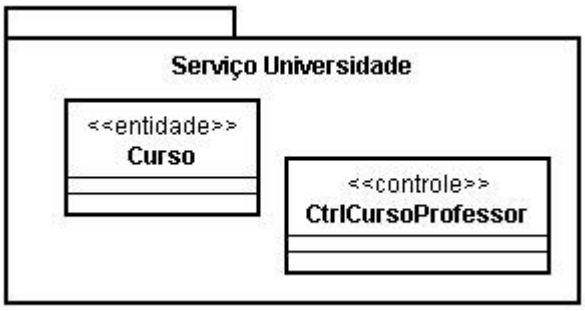
O principal diagrama de classes na visão lógica do modelo é normalmente uma figura dos packages do sistema. Cada package também tem o seu próprio diagrama principal, o qual normalmente mostra as classes públicas do package. Outros diagramas podem ser criados de acordo com a necessidade. Alguns dos usos típicos de outros diagramas são:

- Visão de todas as classes de implementação do package;
- Visão da estrutura e comportamento de uma ou mais classes;
- Visão da hierarquia de herança.

Um exemplo do diagrama de classe principal do sistema de matrícula é mostrado na figura abaixo.



O diagrama de classe principal (*Main*) para o package *ServicoUniversidade* é mostrado na figura. Note que a classe *OfertaCurso* não está no diagrama. Esta é uma classe de implementação e foi decidido não mostrá-la no diagrama principal do package.

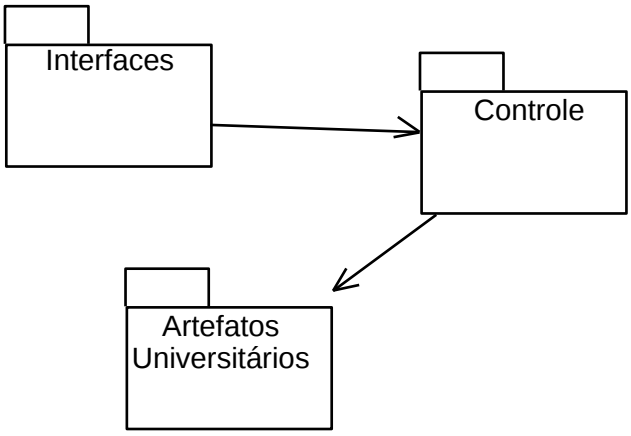


Quando mais packages e classes são adicionados ao modelo, diagramas adicionais podem ser criados quando necessários.

Um package na visão lógica do modelo é uma coleção packages e/ou classes relacionadas. Agrupando classes em packages, nós podemos olhar ao mais alto nível de visão do modelo (i.e., os packages), ou nós podemos ir fundo no modelo olhando o que está contido dentro dos packages.

Relacionamentos entre Pacotes

Pacotes são relacionados entre si usando um relacionamento de dependência
Se uma classe em um pacote “conversa” com uma classe em outro pacote então um relacionamento de dependência á adicionado ao nível de pacote
Diagramas de Cenário e diagramas de classe são avaliados para determinar relacionamentos entre pacotes



12.4. ATIVIDADE

Construa um Diagrama de Classes inicial para controlar empréstimos e reservas de publicações em uma biblioteca, considerando o seguinte:

- A biblioteca pertence a uma empresa e somente os funcionários da empresa estão aptos a reservar ou emprestar publicações.
- A biblioteca tem acesso aos dados cadastrais dos funcionários.
- Uma publicação pode ser um livro, uma revista, um jornal, etc. A biblioteca pode possuir vários exemplares de uma mesma publicação.
- Qualquer tipo de publicação pode ser emprestada ou reservada.
- Não há número limite de reservas por publicação.
- Quando uma publicação for devolvida e para ela existirem reservas, o primeiro da fila deve ser avisado.
- A reserva expira um dia após o aviso da disponibilidade da publicação ao interessado. Caso o mesmo não venha a emprestar a publicação, ela será considerada disponível e o próximo da fila deve ser avisado.
- Uma reserva pode ser excluída a pedido do usuário.
- Empréstimos de publicações não devolvidas 2 dias após o prazo serão consideradas irregulares e um funcionário da biblioteca deve resgatar o livro.

AULA 13 MODELO DE CLASSES, IMPLEMENTAÇÃO E MODELO RELACIONAL

13.1. APRESENTAÇÃO

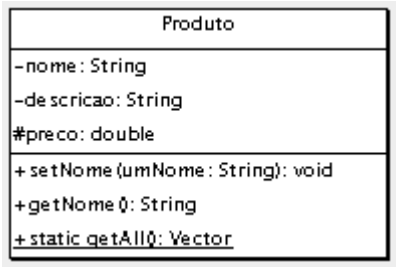
Nesta aula iremos abordar e exercitar que o modelo de classes tem com a codificação em java e com o modelo relacional.

13.2. MAPEANDO DIAGRAMAS DE CLASSE PARA CÓDIGO EM JAVA

Implementar um sistema baseado em um modelo, depende primeiro do paradigma usado na construção do modelo e no paradigma da linguagem escolhida para o desenvolvimento. É possível construir um sistema a partir de um modelo orientado a objetos usando uma linguagem estruturada e vice-versa. Mas, para isso teremos que fazer diversas adaptações.

No nosso caso iremos apresentar a implementação usando a linguagem Java.

Implementando classes simples:



Este é o código em java para a classe descrita pelo diagrama acima.

```
class Produto {
  private String nome;
  private String descricao;
  protected double preco;
  public void setNome(String umNome)  {}
  public String getNome()   {return null;}
  static public Vector getAll() {return null;}
}
```

Implementando associações:

Para implementar uma associação precisamos colocar um atributo de referência. Em qual classe será colocado este atributo, depende da multiplicidade da associação. Estes atributos ou classes que são criados para este fim não devem ser colocados no diagrama de classes.

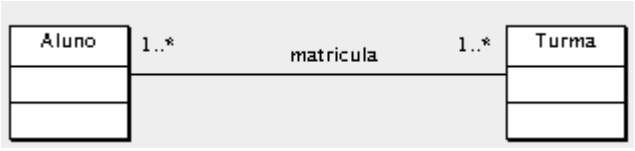
No caso 1 para muitos podemos colocar todo o objeto com multiplicidade 1 como atributo do objeto com multiplicidade muitos:


```
public class ItemdePedido
{
      private int quantidade;
      private EspecificacaoProduto produto;
}
```

ou, se o objeto tiver um código gerado automaticamente na base de dados, apenas o código do objeto com multiplicidade 1 como atributo do objeto com multiplicidade muitos:

```
public class ItemDePedido
{
      private int quantidade;
      private int codProduto;
}
```

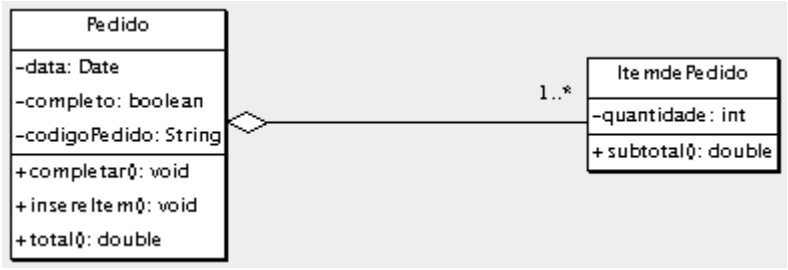
No caso de muitos para muitos:



Existe a necessidade da criação de uma classe para a associação:

```
public class Matricula
{
      private Aluno aluno;
      private Turma turma;
}
```

Agregações:



Uma agregação pode ser implementada usando um objeto para coleções do Java como java.util.Vector. Este objeto é colocado como atributo da classe que agrega e contém objetos da classe agregada;

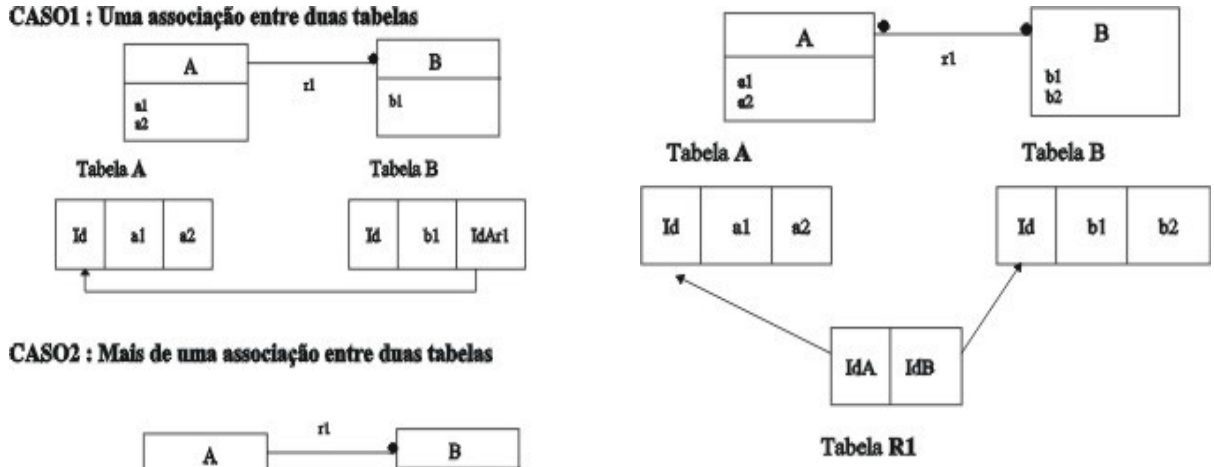
```
public class Pedido
{
      private Date data;
      private String codPedido;
      private boolean completo;
```

```
        private Vector todosItens;
}
```

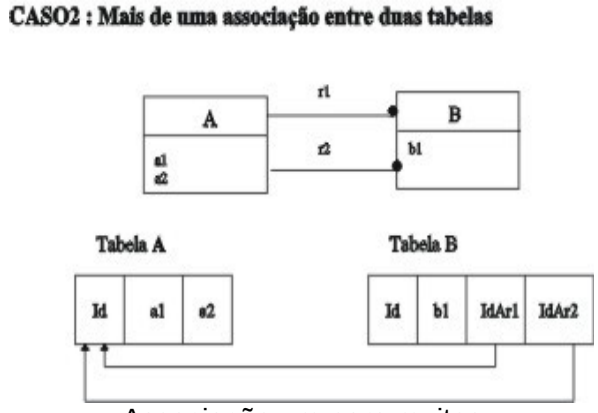
13.3. MAPEANDO CLASSES PARA MODELO RELACIONAL

Associação 1:N
Define-se uma tabela por classe acrescentando-se os atributos identificadores de cada classe à respectiva tabela. A figura abaixo ilustra a situação tratada:

Associação M:N
Define-se uma tabela por classe acrescentando-se os atributos identificadores de cada classe à respectiva tabela. Neste caso, é necessária a criação de uma tabela auxiliar para indicar as associações entre os objetos. A figura abaixo ilustra a situação tratada:



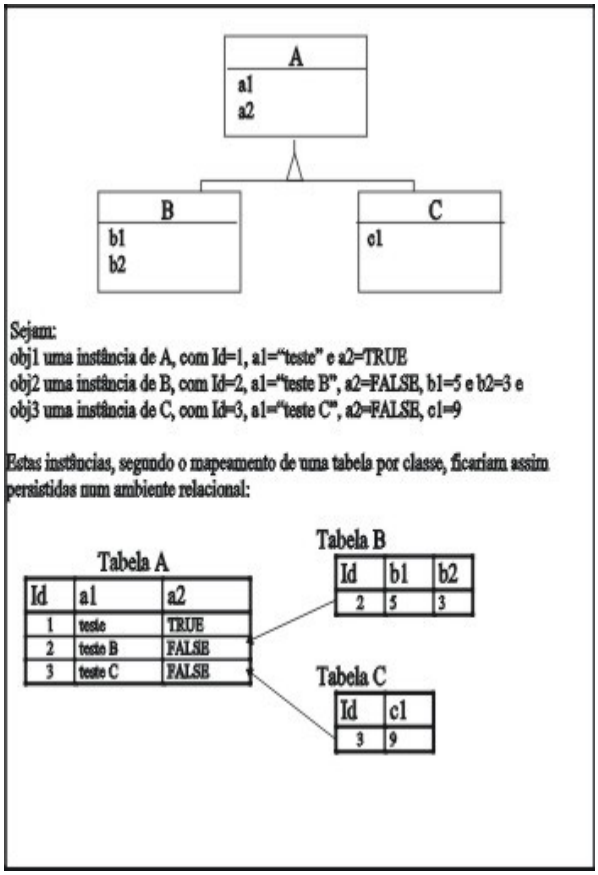
Associação muitos para muitos



Associação um para muitos

Herança
Identificam-se quatro formas para o mapeamento das heranças. São elas:

Uma tabela por classe



Sejam:
obj1 uma instância de A, com Id=1, a1="teste" e a2=TRUE
obj2 uma instância de B, com Id=2, a1="teste B", a2=FALSE, b1=5 e b2=3 e
obj3 uma instância de C, com Id=3, a1="teste C", a2=FALSE, c1=9

Estas instâncias, segundo o mapeamento de uma tabela por classe, ficariam assim persistidas num ambiente relacional:

Tabela A

Id	a1	a2
1	teste	TRUE
2	teste B	FALSE
3	teste C	FALSE

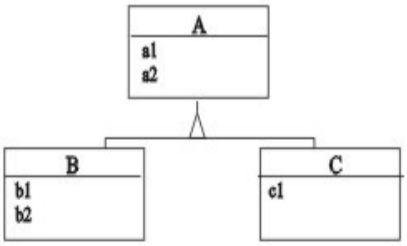
Tabela B

Id	b1	b2
2	5	3

Tabela C

Id	c1
3	9

Uma única tabela



Sejam:
obj1 uma instância de A, com Id=1, a1="teste" e a2=TRUE
obj2 uma instância de B, com Id=2, a1="teste B", a2=FALSE, b1=5 e b2=3 e
obj3 uma instância de C, com Id=3, a1="teste C", a2=FALSE, c1=9

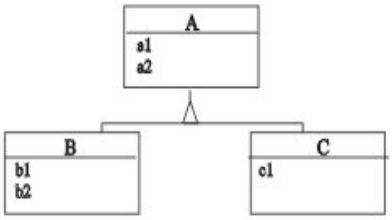
Estas instâncias, segundo o mapeamento em uma tabela, ficariam assim persistidas num ambiente relacional:

Tabela ABC

Id	Tp	a1	a2	b1	b2	c1
1	A	teste	TRUE	null	null	null
2	B	teste B	FALSE	5	3	null
3	C	teste C	FALSE	nul	null	9

onde *tp* indica qual é a classe que a instância pertence e *null* indica a inexistência de dado.

Uso de tabelas apenas para classes específicas



Sejam:
a Classe A uma classe abstrata,
obj2 uma instância de B, com Id=2, a1="teste B", a2=FALSE, b1=5 e b2=3 e
obj3 uma instância de C, com Id=3, a1="teste C", a2=FALSE, c1=9

Estas instâncias, segundo o mapeamento de uma tabela por classe específica, ficariam assim persistidas num ambiente relacional, conform demonstrado nas tabelas abaixo, note que para a recuperação de coleção de instâncias da Classe A, tem-se que fazer uma união das duas tabelas.

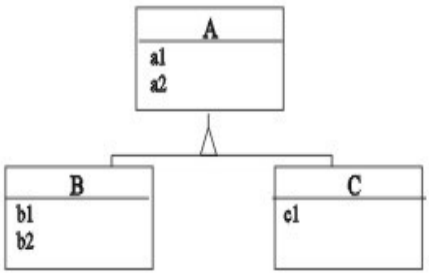
Tabela AB

Id	a1	a2	b1	b2
2	teste B	FALSE	5	3

Tabela AC

Id	a1	a2	c1
3	teste C	FALSE	9

Mista



Sejam:
obj1 uma instância de A, com Id=1, a1="teste" e a2=TRUE
obj2 uma instância de B, com Id=2, a1="teste B", a2=FALSE, b1=5 e b2=3 e
obj3 uma instância de C, com Id=3, a1="teste C", a2=FALSE, c1=9

Estas instâncias, misturando algumas das técnicas acima descritas, poderiam ficar assim persistidas num ambiente relacional:

Tabela AB

Id	a1	a2	b1	b2
1	teste	TRUE	null	null
2	teste B	FALSE	5	3
3	teste C	FALSE	null	null

Tabela C

Id	c1
3	9

13.4. ATIVIDADE

1) Construa um Diagrama de Classes inicial para controlar empréstimos e reservas de publicações em uma biblioteca, considerando o seguinte:

- A biblioteca pertence a uma empresa e somente os funcionários da empresa estão aptos a reservar ou emprestar publicações.
- A biblioteca tem acesso aos dados cadastrais dos funcionários.
- Uma publicação pode ser um livro, uma revista, um jornal, etc. A biblioteca pode possuir vários exemplares de uma mesma publicação.
- Qualquer tipo de publicação pode ser emprestada ou reservada.
- Não há número limite de reservas por publicação.
- Quando uma publicação for devolvida e para ela existirem reservas, o primeiro da fila deve ser avisado.
- A reserva expira um dia após o aviso da disponibilidade da publicação ao interessado. Caso o mesmo não venha a emprestar a publicação, ela será considerada disponível e o próximo da fila deve ser avisado.
- Uma reserva pode ser excluída a pedido do usuário.
- Empréstimos de publicações não devolvidas 2 dias após o prazo serão consideradas irregulares e um funcionário da biblioteca deve resgatar o livro.

2) Crie o código Java para três das classes identificadas acima, que possuas relacionamento.

3) Crie o script SQL de criação dessas 3 classes.

AULA 14 DIAGRAMAS DE INTERAÇÃO

14.1. APRESENTAÇÃO

Nesta aula vamos apresentar os diagramas de seqüência e comunicação

14.2. DIAGRAMAS DE INTERAÇÃO

Os diagramas de Interação, compostos por: Diagramas de Seqüência e Diagramas de Comunicação.

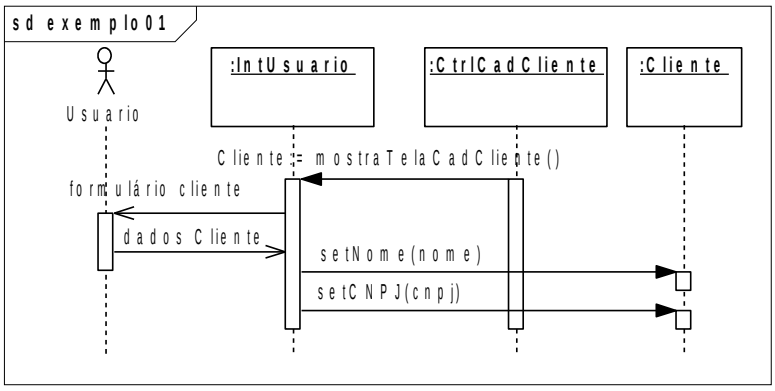
DIAGRAMA DE SEQÜÊNCIA:

- O próximo passo é identificar os objetos envolvidos nele, formando assim diagramas de seqüência.
- Mostra como os objetos se interagem através do tempo em um determinado cenário.
- As linhas verticais são objetos e não classes.
- Podem existir eventos simultâneos.
- Podem existir eventos reflexivos.

PASSOS:

1. Identifique os "principais objetivos do seu sistema".
2. Descreva o cenário.
3. Identifique os objeto origem e o objeto destino.
4. Construa o Diagrama.

Exemplo de um diagrama seqüência na UML:



Os diagramas de seqüência podem representar vários aspectos da interação entre 2 objetos:

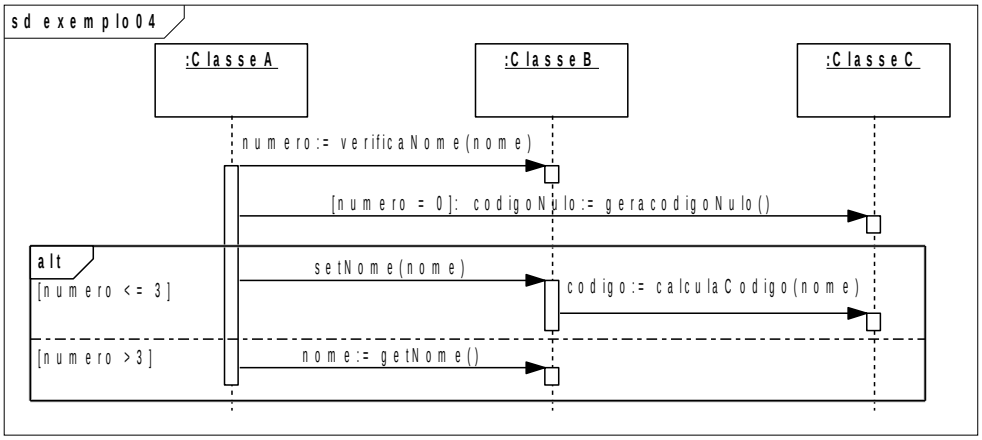
a) Dois tipos de mensagens:
mensagens síncronas (esperam um retorno):

- chamadas de método;
- chamadas de criação de objetos;
- chamadas de destruição de objetos.
- retorno de mensagem;

mensagens assíncronas (não esperam um retorno)

b)Comandos de decisão (if, if else):

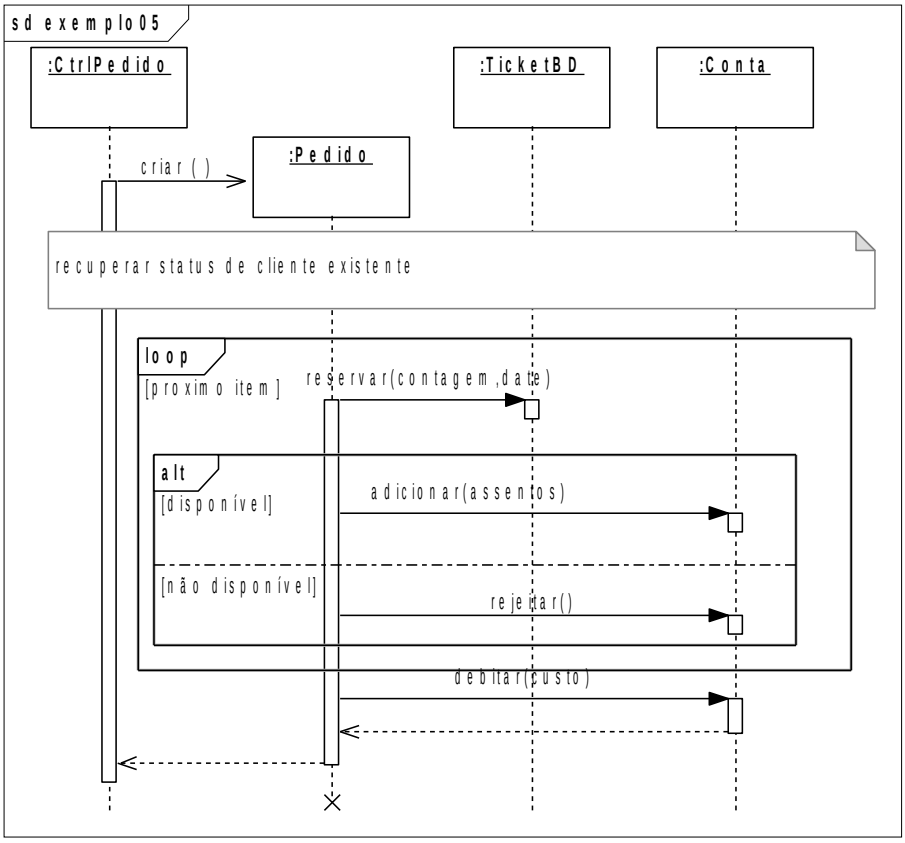
podem ser representados na mensagem ou através de um fragmento combinado (UML 2.0)



c) Repetições (loops):

representados direto na mensagem ou através de fragmentos combinados (UML 2.0)

d) Chamadas a outros diagramas de seqüência (UML 2.0)



Alguns aspectos dependentes da linguagem para expressar a mensagem, podem ser omitidos na fase de análise.

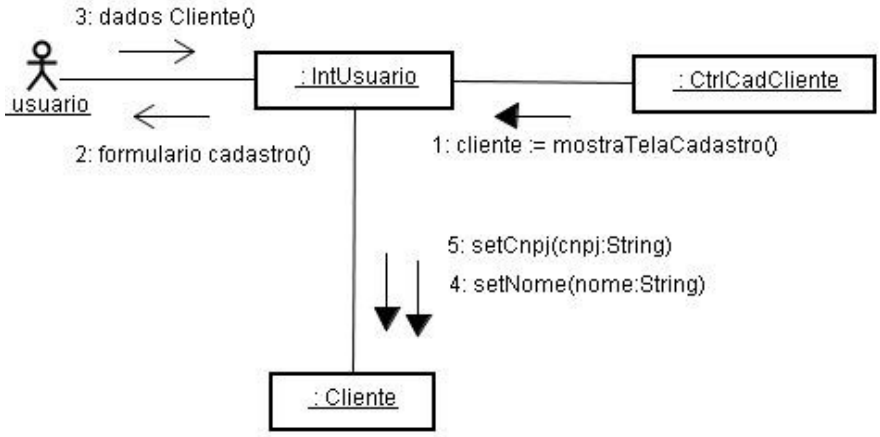
e)Podemos especificar objetos que são criados em tempo de execução do sistema.

f) Podemos também expressar mensagens reflexivas.

g) Também podemos expressar noção de tempo de transmissão de mensagem.

DIAGRAMA DE COMUNICAÇÃO:

- Vieram a substituir os diagramas de colaboração na UML 2.0
- Mesmo comportamento do diagrama de rastreamento de eventos, porém com uma outra visão.
- Algumas ferramentas CASE fazem a transformação automática.
- É a mesma coisa "escrita" de outra forma.



14.3. ATIVIDADE

construa o diagrama de seqüência para os seguintes enunciados:

2.2. Fluxo de Eventos
2.2.1. Fluxo básico

1. O caso de uso começa quando o funcionário seleciona a opção manter dados do ponto. O sistema mostra uma lista com os últimos registro de ponto e as opções "alterar " e "inserir".
2. Se o funcionário seleciona a opção "incluir", o sub-fluxo incluir ponto é executado; Se o funcionário seleciona a opção "alterar", o sub-fluxo alterar ponto é executado;

2.2.2. Subfluxo inserir ponto

1. O sistema mostra uma tela com os campos do ponto, que são: "hora de início", "hora de término", "data", "número do projeto".
2. O funcionário preenche os campos. O sistema mostra os dados preenchidos e pede confirmação
3. O funcionário confirma a inclusão. O sistema mostra uma mensagem de sucesso.

2.2.3. Subfluxo alterar ponto

1. O sistema mostra uma lista com os últimos registro de ponto.
2. O funcionário seleciona o ponto a ser alterado. O sistema mostra uma tela com os dados do ponto, que são: "hora de início", "hora de término", "data", "número do projeto".
3. O funcionário altera os dados necessários. O sistema mostra os dados alterados e pede confirmação
4. O funcionário confirma a alteração. O sistema mostra uma mensagem de sucesso.

AULA 15 DIAGRAMAS DE SEQÜÊNCIA

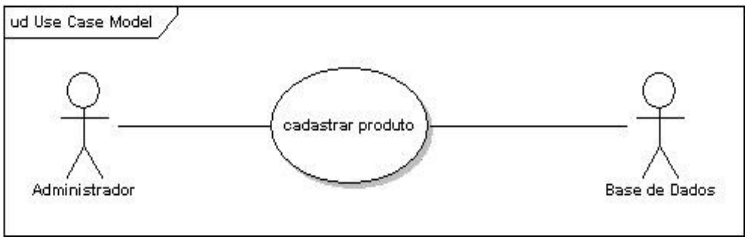
15.1. APRESENTAÇÃO

Construindo e implementando diagramas de sequência em java com jsp e servlet de controle

15.2. DESENVOLVIMENTO

Dicas para a identificação de classes que participam do caso de uso:

- Para cada ator que participa do caso de uso, identifique pelo menos uma classe <<limite>>;
- Para cada caso de uso identifique uma classes <<controle>>;
- Para cada grupo de informações manipulado no caso de uso, identifique uma classe <<entidade>>.



Para o caso de uso cadastrar produto identificamos as seguintes classes:

<<limite>> IntAdministrador Repositorio	<<controle>> CtlrCadastrarProduto	<<entidade>> Produto

Agora vamos desenhar o diagrama com base nos eventos.

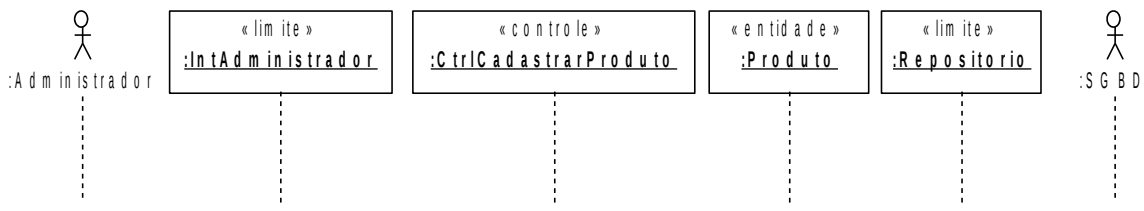
2.2. Fluxo de Eventos
2.2.1. Fluxo básico

1. O caso de uso começa quando o administrador seleciona a opção cadastrar produto. O sistema mostra um formulário com os campos para os dados do produto.

2. O Administrador preenche os dados do produto e seleciona a opção gravar produto. O sistema mostra uma tela com os dados preenchidos do produto e pede confirmação;

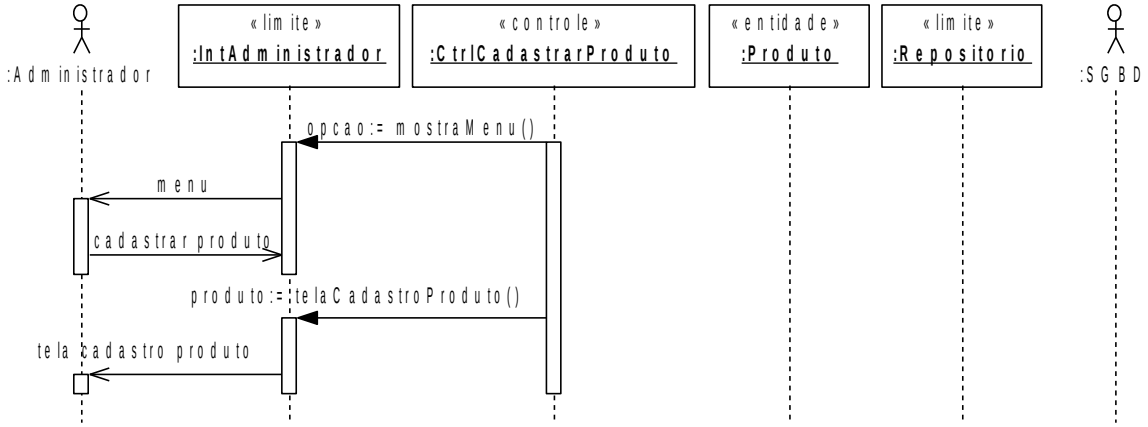
3. O Administrador seleciona confirmar. O sistema grava as informações do produto na base e mostra uma tela confirmando o sucesso da operação.

Primeiro vamos colocar as instâncias (objetos) das classes identificadas acima no diagrama:



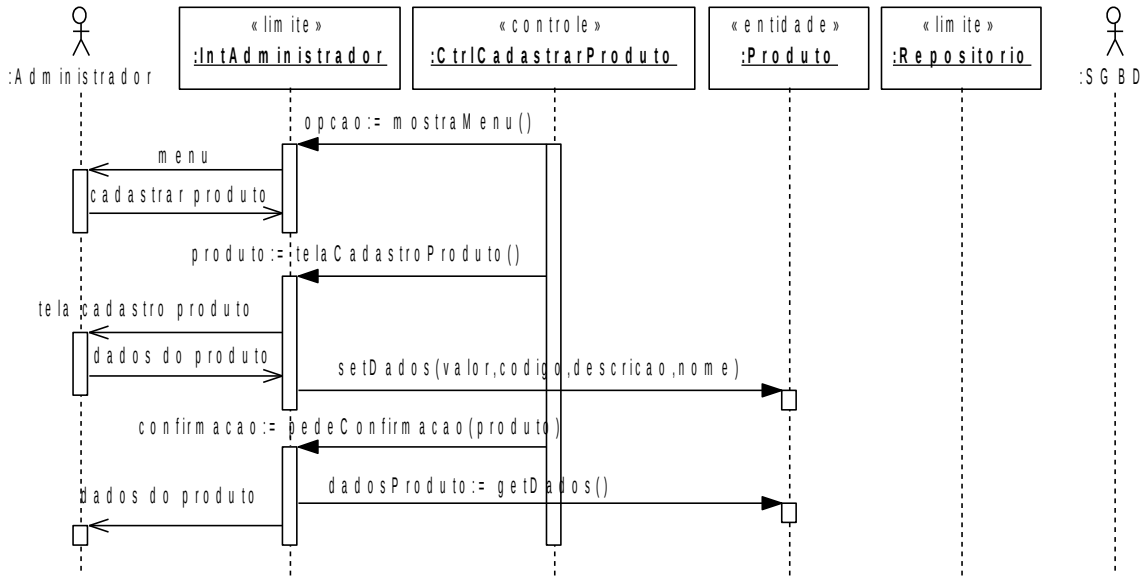
Agora vamos colocar as mensagens relativas ao primeiro evento do fluxo básico:

1. O caso de uso começa quando o administrador seleciona a opção cadastrar produto. O sistema mostra um formulário com os campos para os dados do produto.



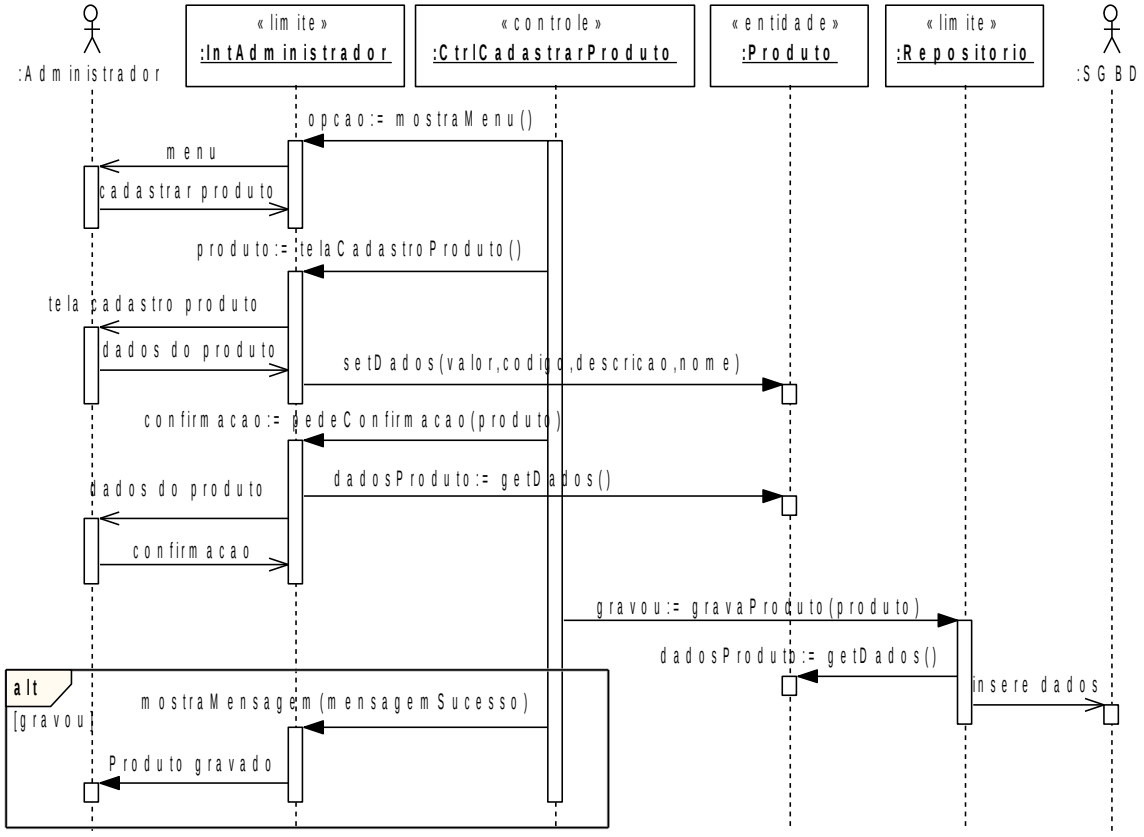
Agora vamos colocar as mensagens relativas ao segundo evento do fluxo básico:

2. O Administrador preenche os dados do produto e seleciona a opção gravar produto. O sistema mostra uma tela com os dados preenchidos do produto e pede confirmação;



Agora vamos colocar as mensagens relativas ao segundo evento do fluxo básico:

3. O Administrador seleciona confirmar. O sistema grava as informações do produto na base e mostra uma tela confirmando o sucesso da operação.



15.3. ATIVIDADE

1. Construa o digrama de seqüência, e implemente o caso de uso “alterar Produto” para o mesmo sistema.

AULA 16 DIAGRAMAS DE ESTADOS

16.1. APRESENTAÇÃO

Vimos e estudamos os diagramas de casos de uso, diagramas de classes e diagramas de interação. Em continuação vamos apresentar, a partir desta aula, os diagramas de estados.

16.2. DIAGRAMAS DE ESTADOS:

São diagramas que representam os estados que um objeto pode assumir e as mudanças de um estado para outro. São compostos basicamente por: Estados e Transições:

Estados:

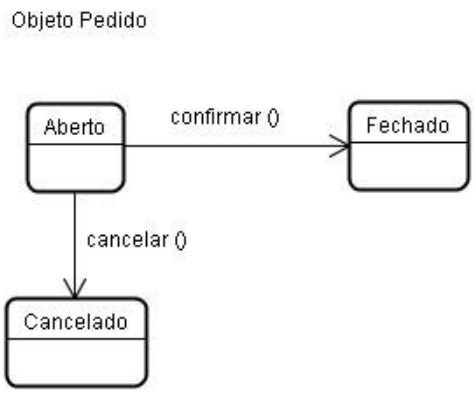
Condição ou situação na vida do objeto
Notação:

Um Estado pode ter:

- ações de entrada/saida
- transições internas
- subestados

Transições

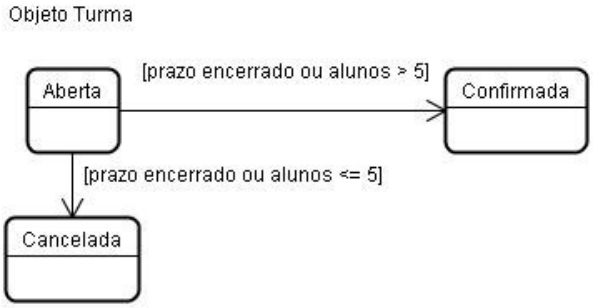
Representam a mudança de um estado para outro



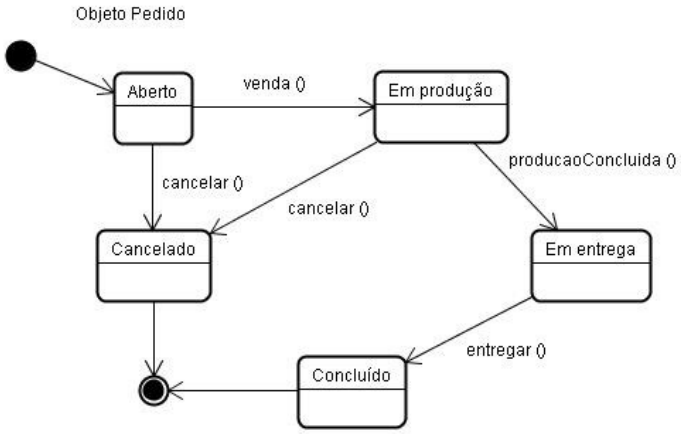
As transições podem ter:

- Estado Origem
- Estado Destino

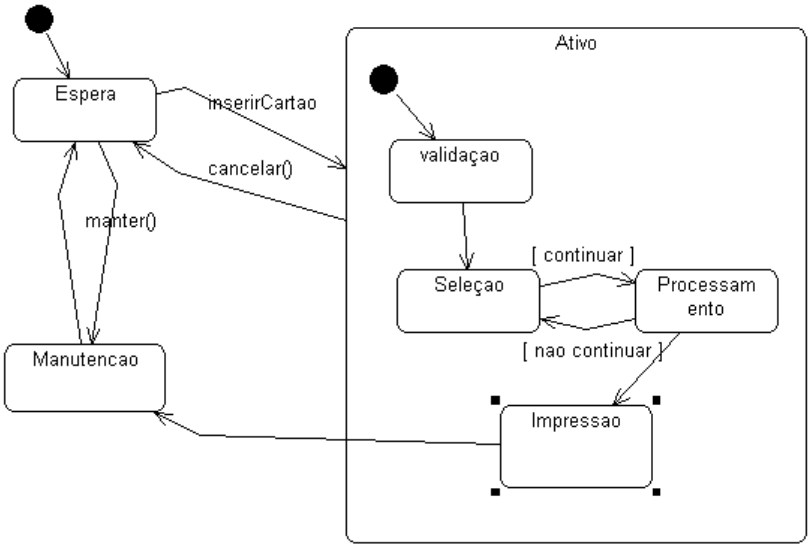
- Evento
- Condições de proteção
- Ações



Exemplo:
Um diagrama que representa os estados de um pedido em uma empresa que fabrica itens sob medida. O pedido é aberto. Uma vez efetuada a venda, o pedido passa a para a produção. Terminada a produção, o pedido passa para a entrega e, uma vez entregue ele é concluído. O pedido só pode ser cancelado enquanto está aberto ou em produção.



Subestados:



16.3. ATIVIDADE

1) Fazer um diagrama de estados para representar os diferentes estados de um funcionário em uma empresa.
2) Fazer um diagrama de estado para representar os diferentes estados de uma fita de vídeo em uma vídeo-locadora.

AULA 17 DIAGRAMAS DE COMPONENTES E DE IMPLANTAÇÃO

17.1. APRESENTAÇÃO

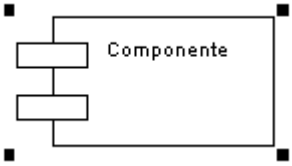
Diagramas de componentes e de Implantação

17.2. DIAGRAMAS DE COMPONENTES

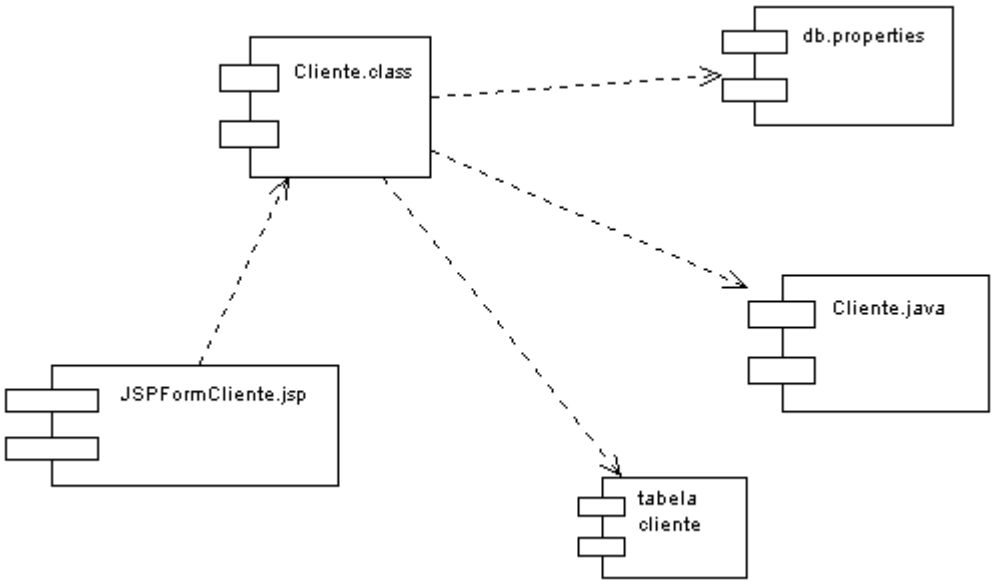
Mostram as dependências entre os componentes do sistema.

Componentes: "Arquivos" que o sistema necessita para funcionar.

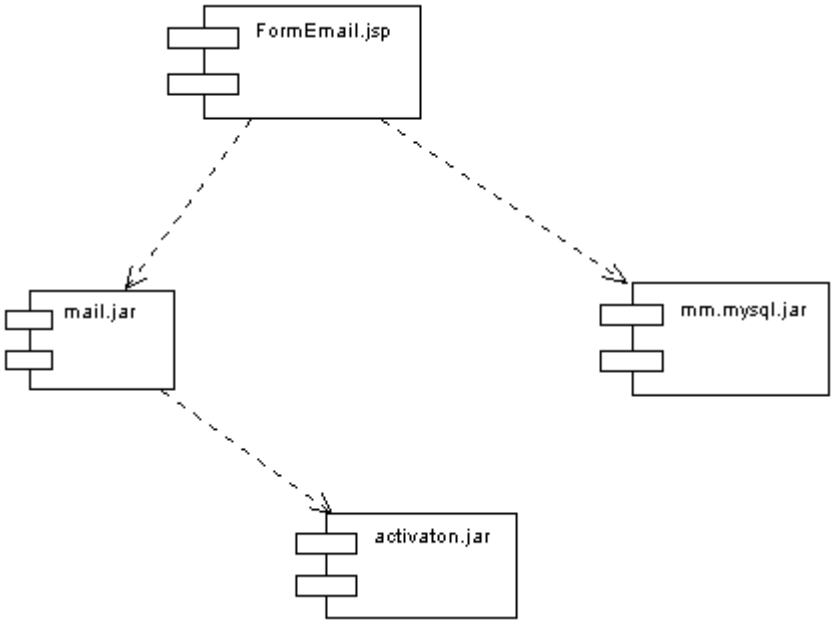
Notação:



As dependencias entre os arquivos são representadas como no exemplo:



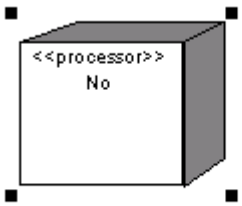
Os componentes podem ser arquivos de propriedades, arquivos fonte, arquivos html, jsp e até mesmo bibliotecas que precisam ser instaladas.



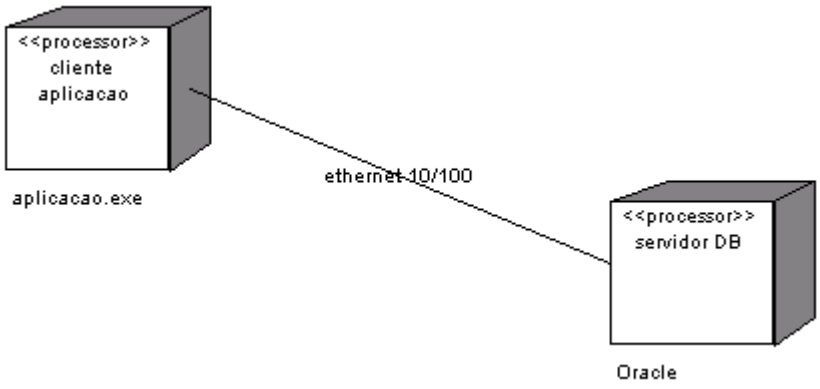
17.3. DIAGRAMAS DE IMPLANTAÇÃO

São diagramas que representam a arquitetura de hardware e software do sistema

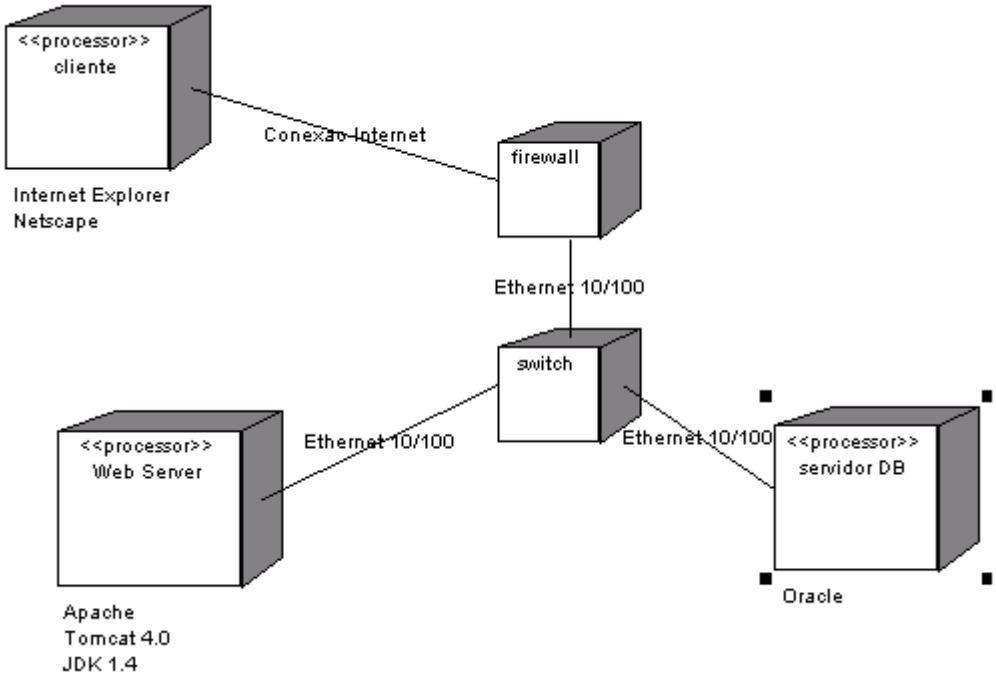
Nós: representam as máquinas e o software que roda em cada uma
Notação:



Ligações: representam as conexões físicas entre as máquinas:



Exemplo:



17.4. ATIVIDADE

Faça um diagrama de implantação para o problema do sistema de Matrículas

BIBLIOGRAFIA BÁSICA

BECK, K; CUNNINGHAM, W.. A laboratory for teaching Object-Oriented thinking. *Anais do International Conference on Object-Oriented Programming, Systems, Languages and Applications OOPLSA89*. New Orleans, EUA: 1989.

BEZZERRA, E.. *Princípios de análise e projeto de sistemas com UML*. Rio de Janeiro, Brasil: Campus, 2002.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.. *The Unified Modeling Language user guide*. 2a. ed.Westford, Massachusets, EUA: Addison-Wesley, 2005.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J.. *The Unified Software Development Process*. Reading, Massachusetts, EUA: Addison-Wesley, 1999.

LARMAN, C.. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3a. ed. New Jersey, EUA: Addison-Wesley Professional, 2004.

OMG. *Unified Modeling Language - Superstructure specification Versão 2.0*. OBJECT MANAGEMENT GROUP: Nedham, Massachusets, EUA, 2005

PRESSMAN, R. S.. *Software Engineering: a practitioner's approach*. 6a. ed .New York, EUA: McGraw-Hill, 2005.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G.. *The unified modeling language reference manual*. 2a. ed. Reading, Massachusetts, EUA: Addison-Wesley, 2004.