Análise e Projeto Orientado a Objetos utilizando UML 2.0 (Unified Modeling Language)

Prof. Flávio de Oliveira Silva, Ph.D.
flavio@facom.ufu.br

Sumário

- Engenharia de Software
  - Visão Geral
  - Ciclo de Vida do Software
- Orientação a Objetos – Conceitos
- Análise Orientada a Objetos
- UML
  - Visão Geral
  - Itens Estruturais e Comportamentais
  - Relacionamentos
  - Diagramas

Indústria de Software - Histórico

- A indústria de Software está em constante evolução
  - Década de 1960
    - Orientação Batch
    - Distribuição limitada
    - Software customizado
  - Década de 1970
    - Multiusuário
    - Tempo real
    - Bancos de Dados
    - Produto de Software

Indústria de Software - Histórico

- Década de 1980
  - Sistemas distribuídos
  - “Inteligência” embutida
  - Hardware Acessível (PCs)
  - Impacto de consumo
- Década de 1990 – Atual
  - Sistemas desktop poderosos
  - Tecnologias OO (Orientada a Objetos)
  - Sistemas Especialistas
  - Redes Neurais Artificiais
  - Computação Paralela

Panorama Atual

- Software
  - Maiores funcionalidades
  - Maior complexidade
  - Abrangência de um maior número usuários
  - Especialização do trabalho, exigindo a participação de equipes em seu desenvolvimento
  - Sistemas distribuídos baseados na WEB
  - Mercado competitivo

Características do Software

- Software é desenvolvido e não produzido no sentido clássico (industrial)
  - Custo de Software é na engenharia e não na reprodução
- Software não se 'gasta'
  - Custos em sua manutenção
- Software precisa se adaptar a novas tecnologias

A "Crise" do Software

- Dificuldades no Trabalho com Software
  - Medidas pobres de eficiência e qualidade
- Insatisfação do usuário é freqüente
  - Pouco entendimento dos requisitos
  - Problemas de Comunicação entre o usuário e o analista
- A qualidade do software é freqüentemente suspeita
  - Poucas medidas e critérios de qualidade
- Software existente é muito difícil de manter
  - E tem que ser mantido até ser substituído

A "Crise" do Software - Causas

- Introdução de erros no processo
  - Má especificação
  - Mau projeto
  - Má implementação
  - Testes incompletos ou mal feitos
- Problemas de comunicação homem-máquina (Entendimento da lógica do computador)
- Problemas de Gerência
  - Falta de treinamento em novas técnicas de desenvolvimento
  - O processo está evoluindo muito rapidamente em função do aprendizado. Necessidade de reciclagem.

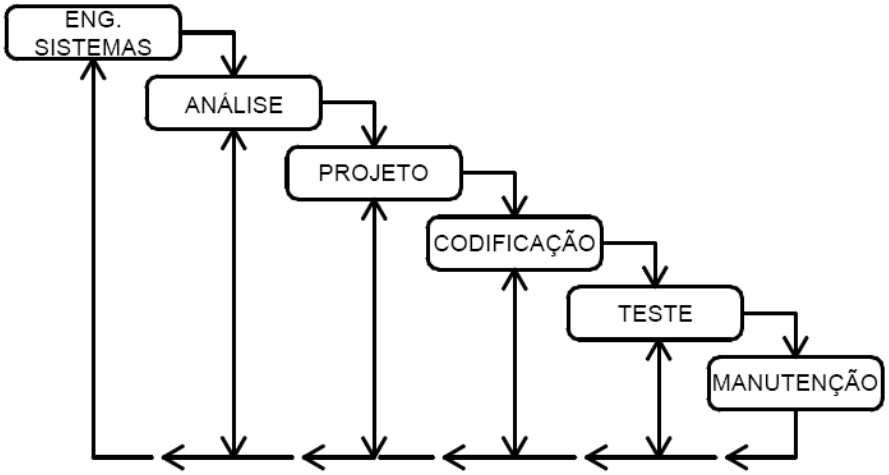
Engenharia de Software

- Estudo e aplicação de Métodos e Técnicas com o objetivo de tornar o desenvolvimento de software mais “eficiente”
- “O estabelecimento e uso de princípios de engenharia de forma a obter economicamente software confiável e que funcione eficientemente em máquinas reais.”
- Existe como disciplina há pouco tempo
- Estabelece um diferencial entre um engenheiro de software profissional e um “praticante da informática”
- Novos profissionais são agentes de mudanças (ou de
- problemas…)
  - Oportunidades, Desafios e Perigos...

Engenharia de Software

- Para o desenvolvimento de software uma linguagem de modelagem não é suficiente
- Precisamos também de um processo de desenvolvimento:
  - Linguagem de modelagem + processo de desenvolvimento = método (ou metodologia) de desenvolvimento
- O processo de desenvolvimento define quem faz o que, quando e como, para atingir os objetivos necessários.

Ciclo de Vida - Modelo Cascata



Engenharia de Software

- **ENGENHARIA DE SISTEMAS**
  - Levantamento dos requisitos
  - Inserir o sistema em um contexto maior –Hardware; Pessoas; Outros sistemas
  - Visão geral e ampla do sistema
  - Riscos; Custos; Prazos; Planejamento

Engenharia de Software

- **ANÁLISE**
  - Continua o processo de coleta de requisitos, porém concentra-se no âmbito do software
  - Modelos – Dados; Funções e comportamentos
  - Particionamento do problema
  - Documentação e Revisão dos requisitos
    - ANÁLISE ESTRUTURADA – DFD
    - ANÁLISE ORIENTADA A OBJETOS – Diagramas de Caso Uso

Engenharia de Software

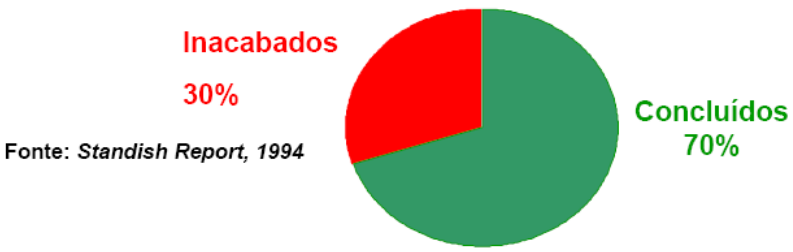
- **PROJETO**
  - "Como" o software irá executar os requisitos
  - Estrutura de dados; Arquitetura do Software;
  - Detalhes de execução; caracterização da
  - interface
  - Produzir um modelo que permita a sua
  - construção posterior
    - PROJETO ESTRUTURADO – Módulos
    - PROJETO ORIENTADO A OBJETOS – Atributos;Especificação dos Métodos; Mensagens
    - Diagramas de Sequência; Diagrama de Classes

Engenharia de Software

- **CODIFICAÇÃO**
  - "Traduzir" o projeto para uma linguagem de
  - computador
  - Projeto detalhado pode levar a uma codificação
  - mecânica (Ferramenta CASE)
- **TESTES**
  - Verificação se o código atende aos requisitos
  - Aspectos lógicos e internos do software – Teste de todas as instruções
  - Aspectos funcionais externos – entrada produz o resultado esperado

Modelo Cascata - Crítica

- Resultado de projetos de software realizados nos EUA no início da década de 90 (Todos baseados no modelo Cascata)
  - 53% custaram até 200% acima da estimativa inicial.
  - Estimou-se que US$ 81 bilhões foram gastos em projetos fracassados só no ano de 1995



Fonte: *Standish Report, 1994*

Modelo Cascata – O que deu Errado?

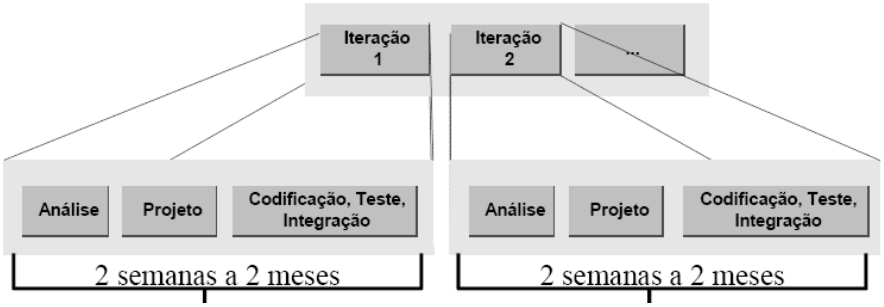
- O modelo cascata é fortemente baseado em suposições oriundas dos processos de engenharia convencionais
- Algumas dessas suposições não foram confirmadas na prática
  - Todos os requisitos podem ser precisamente identificados antes do desenvolvimento
  - Os requisitos são estáveis
  - O projeto pode ser feito totalmente antes da implementação

Modelo Cascata – O que deu Errado?

- **Instabilidade dos Requisitos**
  - O mercado está em mudança constante.
  - As tecnologias inevitavelmente mudam e evoluem
  - A vontade e objetivos dos usuários mudam, muitas
  - vezes, de forma imprevisível.
- **Projeto Completo antes da Implementação?**
  - Pergunte a qualquer programador!
  - Uma especificação completa tem que ser tão detalhada quanto o próprio código
  - Desenvolver software é uma atividade intrinsecamente "difícil"

Ciclo de Vida - Modelo Iterativo

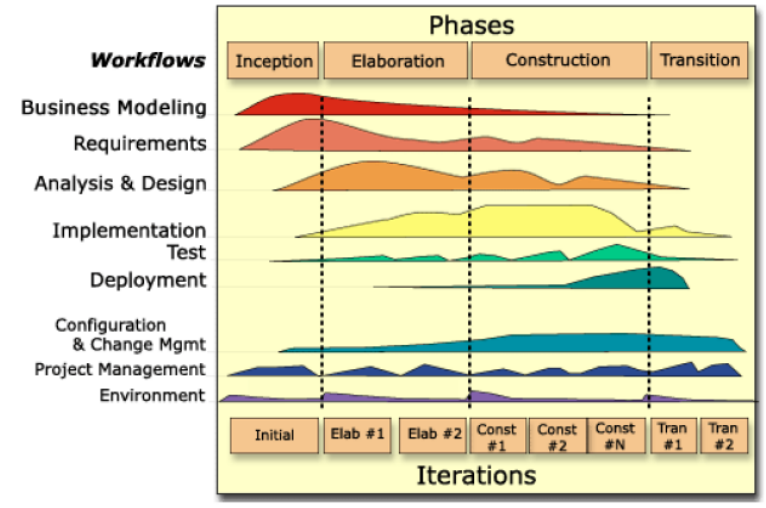
- Passos curtos, feedback e refinamento
- Iterativo, incremental, com intervalos de tempo (ciclos) pré-estabelecidos



Modelo Iterativo

- Baseia-se no fato de que não se deve ter o software inteiro funcionando por inteiro no primeiro release. Isto é um grande risco!
- Um processo de desenvolvimento deve ser:
  - Iterativo - Ter várias iterações no tempo. A iteração dura entre 2 semanas e 2 meses
  - Incremental - Gerar novas versões incrementadas a cada release.
- A cada iteração aumenta a compreensão do problema e são introduzidos aperfeiçoamentos sucessivos.

Modelo Iterativo - RUP

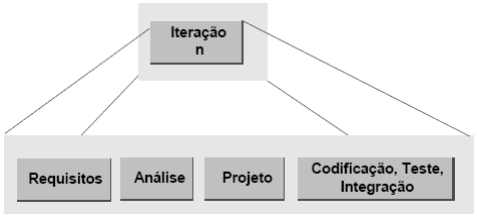


Modelo Iterativo - RUP

- No RUP (Rational Unifed Process) a ênfase é dada na criação de MODELOS (UML) ao invés de documentos.
- As atividades de desenvolvimento são orientadas por caso
- de uso.
- O RUP encoraja o controle de qualidade e o gerenciamento de riscos, contínuos e objetivos
- O desenvolvimento é divido em **FASES e ITERAÇÕES.**
  - FASE – Período de tempo entre marcos do processo, onde um conjunto bem definido de objetivos é alcançado.
  - ITERAÇÃO – Em cada fase, ocorrem várias iterações

Modelo Iterativo - RUP

- Em casa FASE acontece várias **ITERAÇÕES**
  - Uma iteração equivale a um ciclo completo de desenvolvimento
  - Cada iteração resulta em um projeto executável
  - Ao final de cada iteração é possível avaliar se as metas foram alcançadas e caso seja necessário é possível reestruturar o projeto



Modelo Iterativo – RUP - Fases

- **Concepção**
  - Estabelece os casos de negócio para o projeto e delimita o escopo do projeto.
  - Os Casos de negócio incluem: Critérios de Sucesso; Avaliação de Riscos; Recursos Necessários. Durante a concepção é comum a criação de um protótipo executável, utilizado como testes para concepção.
  - Ao final desta fase deve ser feita a decisão de continuar ou não o desenvolvimento
- **Elaboração**
  - Suas metas incluem: Análise do problema;Estabelecimento de uma arquitetura sólida; Eliminação de elementos de mais alto risco.
  - É necessário a maioria dos requisitos do sistema.
  - A implementação deve mostrar escolha da arquitetura.

Modelo Iterativo – RUP - Fases

- **Construção**
  - Desenvolvimento de maneira iterativa e incremental um produto completo.
  - Descrevendo os requisitos restantes e critérios de aceitação. Nesta fase o sistema ganha corpo, conclui-se a implementação e a realização de testes do software.
  - Ao final desta fase deve ser decidido se o software,ambientes e usuários estão prontos para se tornarem operacionais.
- **Transição**
  - Fornece o sistema a seus usuários finais.
  - Esta fase é tipicamente iniciada pelo fornecimento de uma versão beta.
  - Nesta fase são feitos desenvolvimentos adicionais a fim de ajustar o software.

RUP – Fluxos de Trabalho

- Modelagem do Negócio
  - Descreve a estrutura e a dinâmica da empresa
- Requisitos
  - Identificação dos requisitos a partir de casos de uso (Use Cases)
- Análise e Projeto
  - Descreve as várias visões da arquitetura através de modelos UML
- Implementação
  - Desenvolvimento do software; Teste Unitários e Integração
- Teste
  - Casos de teste; procedimentos e medidas para acompanhamento de erros
- Implantação
  - Configuração do sistema a ser entregue

RUP – Fluxos de Trabalho

- Gerenciamento da Configuração
  - Controle de modificações
- Gereciamento do Projeto
  - Descreve as estratégias para o trabalho com o processo iterativo
- Ambiente
  - Infra-estrutura necessária para o desenvolvimento do sistema

Por que a Orientação a Objetos?

- As abstrações podem corresponder às “coisas” do domínio do problema, facilitando o entendimento
- Esta abstração facilita a comunicação com os usuários
- Os mesmos objetos existem em todas as fases e uma notação única facilita a INTEGRAÇÃO ENTRE FASES de desenvolvimento
- A tecnologia de objetos facilita o entendimento do domínio do problema, permitindo o GERENCIAMENTO DA COMPLEXIDADE através da modularização
- Facilidade de mudanças através do ENCAPSULAMENTO de dados

Por que a Orientação a Objetos?

- Capacidade de aproveitar novas plataformas e ferramentas
- Facilidade de manutenção
- Economia de custos
- Encapsulamento das aplicações existentes
- Melhores interfaces
- Maior produtividade
- Participação no "futuro da computação“
- Prova da capacidade de usar a tecnologia
- Rápido desenvolvimento de aplicações estratégicas
- Reuso de software

Por que a Orientação a Objetos?

- Domínio do Problema (Mundo Real)

- Domínio da Solução (Software)

Orientação a Objetos - Conceitos

- OBJETO
- MÉTODO
- MENSAGEM
- CLASSE
- CLASSIFICAÇÃO
- GENERALIZAÇÃO
- ESPECIALIZAÇÃO
- HERANÇA
- POLIMORFISMO
- SOBRECARGA
- ENCAPSULAMENTO
- ABSTRAÇÃO
- MODULARIZAÇÃO

OBJETO

- Entidades que possuem **dados e instruções** sobre como manipular estes dados
- Os objetos estão ligado à solução do problema.
  - Software Gráfico – Objetos: Círculos; Linhas; etc.
  - Software BD – Objetos: Tabelas; Linhas; Campos; etc.
  - Software Comercial: Pedidos; Produtos; Clientes; etc.
- Na OO a solução do problema consiste em um primeiro momento estabelecer quais os objetos serão necessários.
- Um objeto representa uma entidade: física, conceitual ou de software

OBJETO

- Os dados mantidos pelo objeto são chamados de atributos(propriedades)
- Os atributos de um objeto representam seu ESTADO, ou seja, o valor de seus atributos em um determinado momento.
- Objetos possuem IDENTIDADE, ou seja, cada objeto é diferente do outro e cada um tem seu próprio tempo de vida
- Cada objeto tem uma identidade única, mesmo que o estado seja idêntico para ambos os objetos

OBJETO

- Dados ligados ao objeto – Exemplos:
  - Círculo – ponto_centro, raio
  - linha – ponto_inicio; ponto_final
  - Cliente – Nome;Data Nascimento; Telefone
  - Telefone – Numero; Modelo; Cor
  - Exemplos:

```
//criação de objetos (sintaxe C++)
Ponto p(3,4);
Circle c(p,5.4);
Cliente pessoa;
```

MÉTODOS

- Métodos são procedimentos que determinam como o objeto se comporta.
- Através dos métodos é possível manipular os dados contidos no objeto.
- Os métodos estão ligados ao comportamento do objeto
- Exemplos
  - Um círculo poderia possuir os métodos:

  draw; move; getArea; getPerimeter; setCenter
  - Um cliente poderia possuir os métodos:

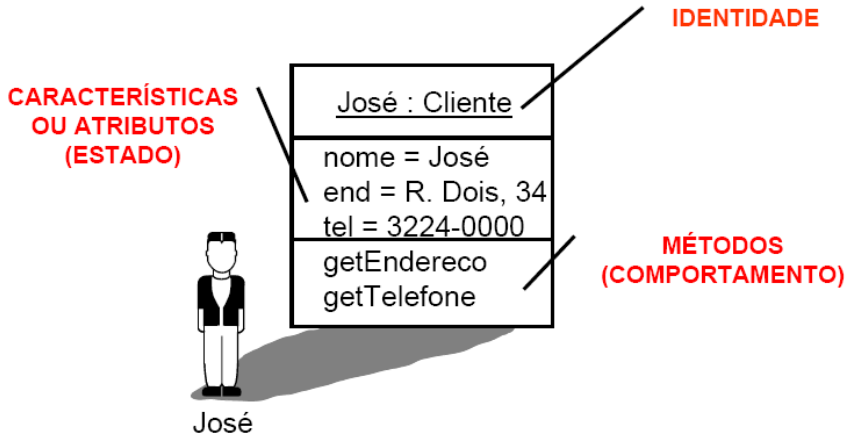
  calculaIdade; getTelefone
  - Um Telefone poderia possui os métodos:

  tocar; discar

MÉTODOS

- Um método é a implementação de uma operação
- Métodos só tem acesso aos dados da classe para a qual foram definidos
- Métodos normalmente possuem argumentos, variáveis locais, valor de retorno, etc.
- Alguns métodos especiais:
  - Construtores – Criam objetos
  - Destrutores – Destroem objetos
  - Acessores – Recuperam o estado de um atributo (getNomeAtributo)
  - Modificadores – Alteram o estado de um atributo (setNomeAtributo)

OBJETOS - RESUMO



MENSAGEM

- Objetos se comunicam entre si através de mensagens.
- Uma mensagem é uma chamada de um método.
- A mensagem possui os seguintes componentes:
  - Receptor – nome do objeto que irá receber a mensagem
  - Método – Método do receptor que será utilizado
  - Argumentos – Informação adicional para a execução do método
  - **Exemplos**

```
Point p(0,0), pNewCenter(2,3);
Circle c(p,3);
c.getArea(); //Exemplo Mensagem
c.setCenter(pNewCenter); //Exemplo Mensagem
```

CLASSE

- Classe é um agrupamento de objetos
- A classe consiste nos métodos e nos dados que um determinado objeto irá possuir.
- Objetos são criados quando uma mensagem solicitando a criação é recebida pela sua classe.
- A programação orientada a objetos consiste em implementar as classes e na utilização das mesmas, através da sua intercomunicação.
- Um objeto é uma instância da classe.
- Os objetos de uma classe compartilham os mesmos atributos, operações, relacionamentos e semânticas
- Objetos só reagem a mensagens que fazem parte das ações do protocolo de sua classe

CLASSIFICAÇÃO

- Na POO classificação consiste em criar classes a partir dos
- objetos envolvidos em um determinado problema
- As classes podem ser criadas a partir do momento em que for possível isolar no domínio do problema objeto que possui atributos e métodos comuns
- Ex: Diferentes tipos de pessoas interagem com uma empresa

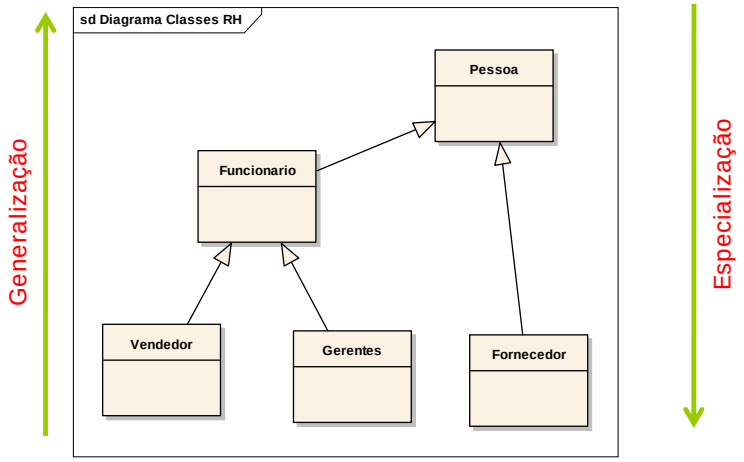
COMPORTAMENTO	CLASSE
Pessoas interessadas nos produtos	???
Pessoas que já compraram os produtos	???
Pessoas que são responsáveis por um grupo de trabalhadores	???
Pessoas responsáveis pela demonstração de produtos e sua venda	???
Trabalhadores da linha de produção	???

Generalização e Especialização

- GENERALIZAÇÃO
  - A generalização consiste em obter similaridades entre as várias classes e partir destas similaridades, novas classes são definidas.
  - Estas classes são chamadas **superclasses**
- ESPECIALIZAÇÃO
  - A especialização por sua vez consiste em observar diferenças entre os objetos de uma mesma classe e dessa forma novas classes são criadas.
  - Estas classes são chamadas **subclasses.**

Generalização e Especialização Exemplo

- Hierarquia de classes



HERANÇA

- Herança é a capacidade de uma subclasse de ter acesso as propriedades da superclasse a ela relacionada.
- Dessa forma as propriedades de uma classe são propagadas de cima para baixo em um diagrama de classes.
- Neste caso dizemos que a subclasse herda as propriedades e métodos da superclasse
- A relação de herança entre duas classes é uma relação da seguinte forma: A “e um tipo de” B, onde A e B são classes.
- Caso esta relação entre as classes não puder ser construída, em geral, também não se tem uma relação de herança entre a classe A a partir da classe B.

HERANÇA

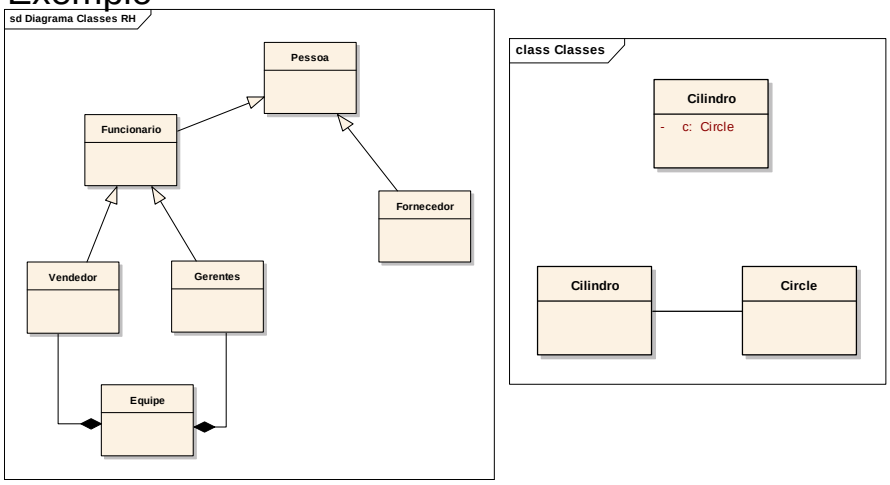
- Exemplos:
  - Um Carro de Passeio “é um tipo de ” veículo; Um caminhão “é um tipo de” veículo;
  - Um círculo “é um tipo de” uma figura geométrica; Um quadrado “é um tipo de” figura geométrica;
  - Um vendedor “é um tipo de” Empregado; Um empregado “é um tipo de” pessoa.
- Herança Múltipla
  - Uma subclasse herda características de mais uma classe
  - Exemplos
    - Um gerente de vendas “é um tipo” de vendedor e também “é um tipo de” gerente;

HERANÇA x USO

- Além da relação de herança entre as classes existe a relação de uso
- HERANÇA
  - classe A “é um tipo de” B
- USO / AGREGAÇÃO (Relação de Conteúdo)
  - classe D “contém” classe C”
  - classe D “usa” classe C”
  - classe C “é parte da” classe D
  - Exemplo: Uma **equipe contém um gerente e** um grupo de **vendedores**

Herança x Uso

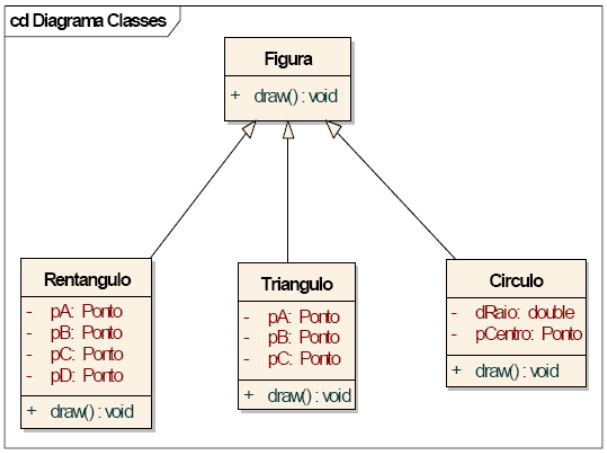
- Exemplo



POLIMORFISMO (Override)

- Os objetos respondem às mensagens que eles recebem através dos métodos.
- A mesma mensagem pode resultar em diferentes resultados. Esta propriedade é chamada de **polimorfismo**
  - **Exemplo: Método getSalario()**
    - Para um empregado qualquer → getsalario() = Salario;
    - Para o gerente → getsalario() = salario + bonificacao;
  - **Exemplo: Método draw()**
    - Para uma figura qualquer desenha uma forma não definida
    - Para o retângulo, triângulo e círculo o mesmo método responde de uma forma diferente

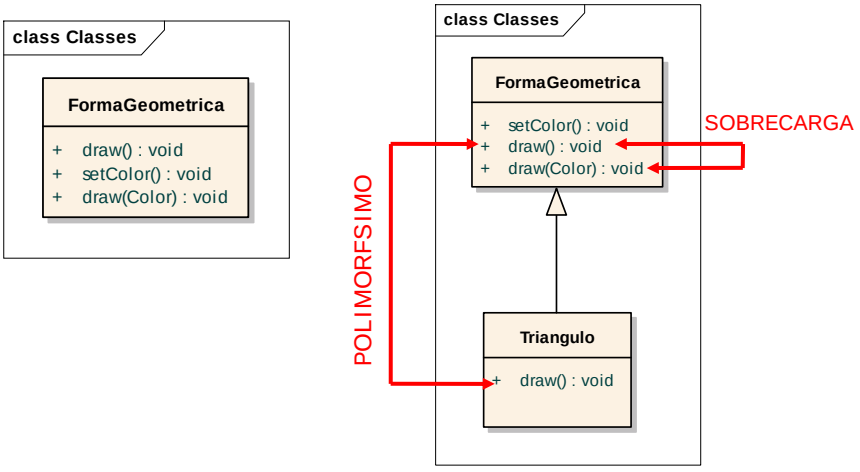
POLIMORFISMO - Exemplo



SOBRECARGA(Overload)

- Nas linguagens orientadas a objetos é possível, em uma classe, a existência de métodos que possuem o mesmo nome, porém com diferentes assinaturas.
- Este conceito é chamado de Sobrecarga (Overload)
  - **Exemplo: Em uma classe Figura, temos os seguintes métodos**
    - draw() → desenha o objeto e utiliza uma cor padrão
    - draw(Color) → desenha o objeto, porém recebe uma cor como parâmetro

SobreCarga - Exemplo



ENCAPSULAMENTO

- Conceito que indica que os dados contidos em um objeto somente poderão ser acessados e/ou modificados através de seus métodos.
- Dessa forma não é possível alterar os dados diretamente, somente através de métodos definidos no objeto
- Exemplo
  - O raio somente pode ser alterado/recuperado pelos métodos setCenter/getCenter.

ENCAPSULAMENTO

- O encapsulamento assegura que toda a comunicação com o objeto seja realizada por um conjunto pré-definido de operações
- O encapsulamento facilita as mudanças, visto que os objetos são isolados uns dos outros, reduzindo desta forma o acoplamento
- Além disso o encapsulamento facilita a manutenção de classes, bem como, garante a integridade dos atributos de um objeto em um determinado instante.

ABSTRAÇÃO

- Abstração é o processo de identificar as qualidades ou propriedades importantes do problema que está sendo modelado.
- Através de um modelo abstrato,pode-se concentrar nas características relevantes e ignorar as irrelevantes.
- Abstração é fruto do raciocínio.
- Através da abstração é possível controlar a complexidade. Isto é feito através da ênfase em características essenciais, fazendo-se uma supressão daquilo que não está ligado ao domínio do problema.

MODULARIZAÇÃO

- Consiste em decompor o problema em partes menores.
- Dessa forma o foco é mantido em itens(classes; pacotes; etc.) menores, coesos e fracamente acoplados.

Praticando os conceitos... Atividade

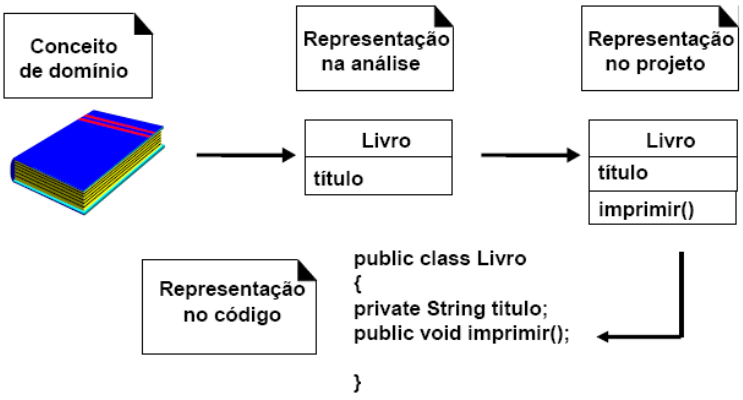
- Utilizando os conceitos
  - Herança
  - Polimorfismo
  - Sobrecarga
  - Classe
  - Atributo
  - Método
- Propor uma modelagem relacionada ao seu ambiente de trabalho

Análise Orientada a Objetos (OOA) Projeto Orientado a Objetos (OOD)

- As técnicas tradicionais (Análise e Projeto Estruturado) não são adequadas para o desenvolvimento de software utilizando a orientação à objetos.
- A utilização do orientação à Objetos solicita uma nova forma de abstrair e entender o problema.
- A linguagem UML é um padrão de diagramação para visualizar os resultados da análise e projeto orientados à objetos

Análise Orientada a Objetos (OOA)
Projeto Orientado a Objetos (OOD)

- Exemplo
  - O conceito “Livro” em um sistema de biblioteca



Análise Orientada a Objetos(OOA)

- Objetivo básico
  - Identificar classes a partir das quais objetos serão representados como instâncias
- Envolve as seguintes tarefas
  - Identificação de Objetos
  - Especificação de Atributos
  - Definição de métodos
  - Comunicações entre objetos

Análise Orientada a Objetos(OOA)

- IDENTIFICAÇÃO DE OBJETOS
  - Entidades externas (Outros sistemas; dispositivos;Pessoas)
  - Coisas ligadas ao domínio do problema (Relatórios;Displays;…)
  - Ocorrências ou Eventos (Conclusão de um movimento;Alarme disparado; Clique do mouse; etc.)
  - Papéis ou funções (Engenheiro; Gerente; Vendedor) desempenhados por pessoas
  - Unidades organizacionais (Grupo; Equipe;…)
  - Lugares (Piso de fábrica; área de descarga)
  - Estruturas (Sensores; veículos de quatro rodas;…)

Análise Orientada a Objetos(OOA)

- IDENTIFICAÇÃO DE OBJETOS – CRITÉRIOS
  1. RETENÇÃO DE INFORMAÇÃO – Objeto deve guardar informação que será utilizada pelo sistema
  2. SERVIÇOS NECESSÁRIOS – Conjunto de operações identificáveis que podem mudar o valor de seus atributos
  3. MÚLTIPLOS ATRIBUTOS – Objeto deve conter mais de um atributo
  4. ATRIBUTOS COMUNS – Conjunto de atributos deve ser aplicado a todos os objetos
  5. OPERAÇÕES COMUNS – Conjunto de operações devem ser aplicáveis a todos os objetos.
  6. REQUISITOS ESSENCIAIS – Entidades externas que aparecem no espaço problema que consomem e/ou produzem informação

Análise Orientada a Objetos(OOA)

- Em uma especificação:
  - NOMES são potenciais objetos (e classes)
  - VERBOS são potenciais métodos
- A regra acima deve ser utilizada apenas como referência.
- O entendimento do contexto, das necessidades do usuário são fundamentais para classificar possíveis objetos e métodos

Análise Orientada a Objetos(OOA) Exemplo Especificação

O software SafeHome possibilita que o dono da casa configure o sistema de segurança quando ele for instalado, monitora todos os sensores ligados ao sistema de segurança e interage com o dono da casa através de um teclado (key pad) e teclas de função contidas no painel de controle do SafeHome.

Durante a instalação o painel de controle é usado para “programar” e configurar o sistema. A cada sensor é atribuído um número e um tipo, uma senha mestra é programada para armar e desarmar o sistema e números telefônicos são introduzidos para serem discados quando ocorrer um evento sensor.

Quando um evento sensor é sentido pelo software, ele dispara um alarme sonoro ligado ao sistema. Após um tempo de espera, que é especificado pelo dono da casa durante as atividades de configuração do sistema, o software disca um número telefônico do serviço de monitoração, oferece informações sobre o local, registrando a natureza do evento que foi detectado. O número será novamente discado a 20 segundos até que a ligação telefônica seja completada.

Todas as interações com o SafeHome são gerenciadas por um subsistema de interação com o usuário, que lê a entrada fornecida através do teclado e das chaves de função, exibe mensagens de prompting e informações sobre o status do sistema no mostrador de cristal líquido (LCD). A interação com o teclado assume a seguinte forma...

Análise Orientada a Objetos(OOA) Exemplo Especificação

O software SafeHome possibilita que o **dono da casa** configure o **sistema de segurança** quando ele for instalado, monitora todos os **sensores** ligados ao **sistema de segurança** e interage com o **dono da casa** através de um **teclado** (key pad) e **teclas de função** contidas no **painel de controle** do SafeHome.

Durante a instalação o **painel de controle** é usado para "programar" e configurar o **sistema**. A cada **sensor** é atribuído um **número** e um **tipo**, uma **senha mestra** é programada para armar e desarmar o **sistema** e **números telefônicos** são introduzidos para serem discados quando ocorrer um **evento sensor**.

Quando um **evento sensor** é sentido pelo software, ele dispara um **alarme sonoro** ligado ao **sistema**. Após um **tempo de espera**, que é especificado pelo **dono da casa** durante as atividades de configuração do sistema, o software disca um **número telefônico** do **serviço de monitoração**, oferece **informações sobre o local**, registrando a **natureza do evento** que foi detectado. O número será novamente discado a 20 segundos até que a ligação telefônica seja completada.

Todas as interações com o SafeHome são gerenciadas por um **subsistema** de interação com o **usuário**, que lê a **entrada** fornecida através do **teclado** e das **chaves de função**, exibe **mensagens** de prompting e informações sobre o status do **sistema** no **mostrador de cristal líquido** (LCD). A interação com o teclado assume a seguinte forma...

Análise Orientada a Objetos(OOA)

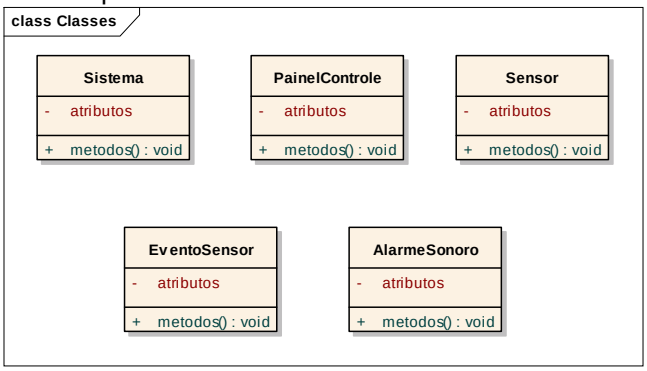
NOME	CRITÉRIO
dono da casa	Papel ou entidade externa
sistema de segurança	Coisa
sensores	Entidade externa
teclado	Entidade externa
teclas de função	Entidade externa
painel de controle	Entidade externa
número	Atributo do sensor
Tipo	Atributo do sensor
senha mestra	Coisa
números telefônicos	Coisa
evento sensor	Ocorrência
alarme sonoro	Entidade externa
tempo de espera	Atributo do sistema
Serviço de monitoração	Unidade Organizacional ou Ent. Externa
informações sobre o local	Atributo do sistema
natureza do evento	Atributo do sistema
subsistema	Entidade externa
entrada	Entidade externa
chaves de função	Entidade externa
mensagens	Entidade externa
mostrador de cristal líquido	Entidade externa

Análise Orientada a Objetos(OOA)

NOME	CRITÉRIO
dono da casa	Rejeitado: 1 e 2 falham embora 6 se aplique
sistema de segurança	Aceito: Todas se aplicam
sensores	Aceito: Todas se aplicam
teclado	Aceito: Todas se aplicam
teclas de função	Aceito: Todas se aplicam
painel de controle	Aceito: Todas se aplicam
número	Rejeitado: 3 falha
Tipo	Rejeitado: 3 falha
senha mestra	Rejeitado: 3 falha
números telefônicos	Rejeitado: 3 falha
evento sensor	Aceito: Todas se aplicam
alarme sonoro	Aceito: 2, 3, 4, 5 e 6
tempo de espera	Rejeitado: 3 falha
Serviço de monitoração	Rejeitado: 1 e 2 falham embora 6 se aplique
informações sobre o local	Rejeitado: 3 falha
natureza do evento	Rejeitado: 3 falha
subsistema	Aceito: Todas se aplicam
entrada	Rejeitado: 3 falha
chaves de função	Aceito: Todas se aplicam
mensagens	Aceito: Todas se aplicam
mostrador de cristal líquido	Aceito: Todas se aplicam

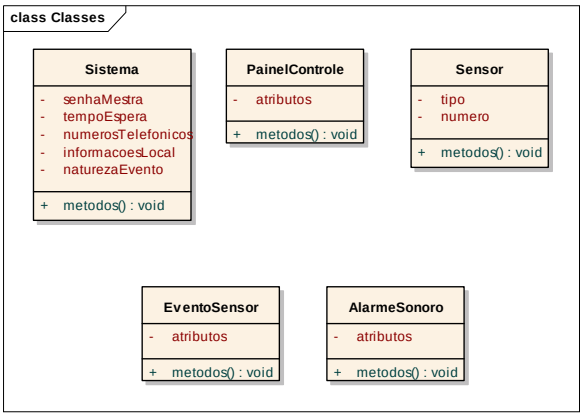
Identificação de Objetos

- Os objetos do painel de controle serão considerados separadamente.
- Os objetos abaixo são o ponto de partida para o desenvolvimento do sistema
- Objetos ainda não possuem atributos e métodos



Especificação de Atributos

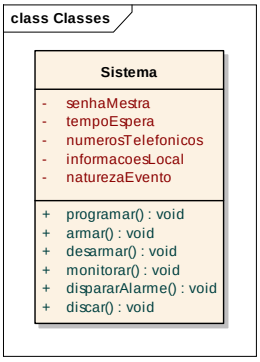
- Necessário estudar e analisar a descrição do sistema
- Pergunta Importante – "Quais informações definem este objeto?"



Definição dos Métodos

- Necessário estudar e analisar a descrição do sistema
- Verbos são potenciais operações

Configure	sentido
Instalado	dispara
Monitora	especificado
Ligados	configuração
Interage	disca
Instalação	registrando
"programar"	detectado
configurar	Discado
programada	ligação
armar	interações
desarmar	gerenciadas
introduzidos	interação
serem discados	lê
ocorrer	fornecida
	exibe



Análise Orientada a Objetos(OOA)

O departamento de obras públicas da cidade de Uberlândia decidiu desenvolver um sistema de computador para rastreamento e conserto de buracos de rua (SIRCOB).

À medida que são registrados buracos de rua, eles recebem um número de identificação e são armazenados de acordo com o endereço da rua, tamanho (numa escala de 0 a 10), localização (no meio dar rua; na calçada; etc.), bairro (determinado a partir do endereço da rua) e prioridade de reparo (determinada a partir do tamanho do buraco).

Dados de ordem de trabalho são associados a cada buraco, e eles incluem localização e tamanho do buraco, número de identificação da equipe de reparos, número de pessoas na equipe, equipamentos designados, horas aplicadas ao reparo, status do trabalho (em andamento, concluído, não concluído), quantidade de material de enchimento usado e custo do reparo (computado a partir das horas trabalhadas, número pessoas, material e equipamentos usados).

Finalmente, um arquivo de danos ocorridos é criado para guardar informações sobre danos registrados devido ao buraco, o qual inclui o nome do cidadão, endereço, número telefônico, tipo de dano e a quantia em reais a ser paga. O SIRCOB é um sistema on-line; as consultas devem ser feitas interativamente.

Análise Orientada a Objetos(OOA)

O departamento de obras públicas da cidade de Uberlândia decidiu desenvolver um **sistema** de computador para rastreamento e conserto de **buracos** de rua (SIRCOB).

À medida que são registrados buracos de rua, eles recebem um **número de identificação** e são armazenados de acordo com **o endereço da rua**, **tamanho** (numa escala de 0 a 10), **localização** (no meio dar rua; na calçada; etc.), **bairro** (determinado a partir do endereço da rua) e **prioridade** de reparo (determinada a partir do tamanho do buraco).

Dados de **ordem de trabalho** são associados a cada buraco, e eles incluem **localização** e **tamanho do buraco**, **número de identificação da equipe de reparos**, **número de pessoas na equipe**, **equipamentos** designados, **horas** aplicadas ao reparo, **status** do trabalho (em andamento, concluído, não concluído), **quantidade** de material de enchimento usado e custo do reparo (computado a partir das horas trabalhadas, número pessoas, material e equipamentos usados).

Finalmente, um **arquivo** de **danos** ocorridos é criado para guardar informações sobre danos registrados devido ao buraco, o qual inclui o **nome do cidadão**, **endereço**, **número telefônico**, **tipo de dano** e a **quantia** em reais a ser paga. O SIRCOB é um sistema on-line; as consultas devem ser feitas interativamente.

Identificação dos Objetos

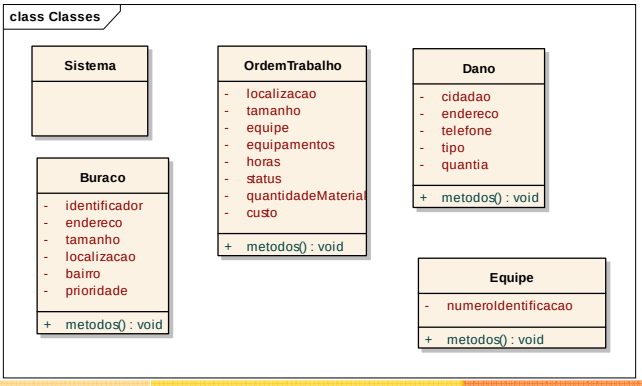
NOMES	CLASSIFICAÇÃO	ANÁLISE
departamento	Unidade Organizacional	Rejeitado: 1 e 2 Falham
sistema	**coisa**	**Aceito: Todas se aplicam**
buracos	**coisa**	**Aceito: Todas se aplicam**
número de identificação	coisa	Rejeitado: 3 falha
endereço da rua	coisa	Rejeitado: 3 falha
tamanho	coisa	Rejeitado: 3 falha
localização	coisa	Rejeitado: 3 falha
bairro	bairro	Rejeitado: 3 falha
prioridade	coisa	Rejeitado: 3 falha
ordem de trabalho	**coisa**	**Aceito: Todas se Aplicam**
número de identificação da equipe	coisa	Rejeitado: 3 falha
número de pessoas	coisa	Rejeitado: 3 falha
equipamentos	coisa	Rejeitado: 3 falha
horas	coisa	Rejeitado: 3 falha
quantidade	coisa	Rejeitado: 3 falha
custo	coisa	Rejeitado: 3 falha
arquivo de danos	**estrutura**	**Aceito: Todas se aplicam**
nome do cidadão	coisa	Rejeitado: 3 falha
Endereço	coisa	Rejeitado: 3 falha
número telefônico	coisa	Rejeitado: 3 falha
tipo de dano	coisa	Rejeitado: 3 falha

Identificação dos Atributos

NOMES	CLASSIFICAÇÃO	ANÁLISE
departamento	Unidade Organizacional	Rejeitado: 1 e 2 Falham
sistema	**coisa**	**Aceito: Todas se aplicam**
buracos	**coisa**	**Aceito: Todas se aplicam**
número de identificação	coisa	Rejeitado: 3 falha (ATRIBUTO)
endereço da rua	coisa	Rejeitado: 3 falha (ATRIBUTO)
tamanho	coisa	Rejeitado: 3 falha (ATRIBUTO)
localização	coisa	Rejeitado: 3 falha (ATRIBUTO)
bairro	bairro	Rejeitado: 3 falha (ATRIBUTO)
prioridade	coisa	Rejeitado: 3 falha (ATRIBUTO)
ordem de trabalho	**coisa**	**Aceito: Todas se Aplicam**
número de identificação da equipe	coisa	Rejeitado: 3 falha (ATRIBUTO)
número de pessoas	coisa	Rejeitado: 3 falha (ATRIBUTO)
equipamentos	coisa	Rejeitado: 3 falha (ATRIBUTO)
horas	coisa	Rejeitado: 3 falha (ATRIBUTO)
quantidade	coisa	Rejeitado: 3 falha (ATRIBUTO)
custo	coisa	Rejeitado: 3 falha (ATRIBUTO)
arquivo de danos	**estrutura**	**Aceito: Todas se aplicam**
nome do cidadão	coisa	Rejeitado: 3 falha (ATRIBUTO)
Endereço	coisa	Rejeitado: 3 falha (ATRIBUTO)
número telefônico	coisa	Rejeitado: 3 falha (ATRIBUTO)
tipo de dano	coisa	Rejeitado: 3 falha (ATRIBUTO)

Identificando Classes e Atributos

- Além dos objetos mostrados a seguir que são identificados diretamente a partir da especificação do sistema outros objetos podem ser necessários dependendo de como o sistema será construído. (Ex.: Equipe; Trabalhador; etc.)



Identificando os Métodos

- Verbos destacados
  - registrados
  - Recebem
  - Armazenados
  - Associados
  - Guardar
  - Consultar
- Outras operações serão necessárias para que o sistema funcione corretamente

UML – Unified Modeling Language

- A UML é a linguagem gráfica para a visualização, especificação, construção e documentação de sistemas de software.
- A notação (a própria UML) é relativamente trivial
- Muito mais importante: habilidade para modelar com objetos
- Só aprender a notação UML não ajuda
- A UML não é
  - um processo ou metodologia
  - APOO
  - regras de projeto

UML – HISTÓRICO

- Linguagens orientadas a objetos surgiram entre a metade da década de 1970 e 1980
- Entre 1989 e 1994 a quantidade de métodos de modelagem aumentou de 10 para 50
- Havia uma dificuldade inicial, por parte dos desenvolvedores, devido a diversidade de métodos, visto que não atendiam completamente às suas necessidades.
- Neste conexto alguns método se destacaram: Notação de Booch; o OOSE (Object-Oriented Software Engineering) de Ivar Jacobson e o OMT (Object Modeling Technique) de James Rumbaugh

UML – HISTÓRICO

- Todos eram completos, apresentando pontos fortes e fracos
- Por volta da metade da década de 1990 Booch (Rational Software), Jacobson (Objectory) e Rumbaugh(GE) se unem para desenvolver a UML
- Outubro/1995 o esboço da versão 0.8 foi lançada
- Junho/1996 – Lançada a versão 0.9 da UML
- Foi estabelecido um consórcio de UML com a participação das empresas: Rational Software; Digital; Hewlett-Packard; I-Logix; Intelicorp; IBM; Icon Computing; MCI SystemHouse; Microsoft; Oracle; Texas Instruments e Unysis.
- Este consórcio apresenta em Janeiro/1997 a UML 1.0

UML – HISTÓRICO

- A UML 1.0 foi oferecida ao OMG (Object Management Group)
- Em 1997 o consórcio se ampliou
- Em Novembro/1997 o OMG adotou a UML 1.1
- O grupo Revision Task Force (RTF) do OMG assume a manutenção da linguagem e em Junho/1998 a versão 1.2 foi lançada
- Em Dezembro/1998 o RTF lançou a versão 1.3 da UML
- 2001 - Versão 1.5
- 2003 – Lançamento da Especificação 2.0 pelo OMG

POR QUE MODELAR?

- Uma empresa bem sucedida é aquela que fornece software de qualidade e é capaz de atender as necessidades dos usuários.
- A modelagem é a parte central de todas as atividades que levam a implantação de um bom software
- Um modelo é uma simplificação da realidade
- Modelos são construídos para:
  - Prover a comunicação entre a estrutura e o comportamento do sistema
  - Visualizar e controlar a arquitetura do sistema
  - Ter uma melhor compreensão do sistema
  - Gerenciar os riscos

POR QUE MODELAR?

- Com a modelagem, quatro objetivos podem ser alcançados:
  - Visualizar o sistema como ele é ou como desejamos que seja
  - Possibilita a especificação da estrutura e o comportamento do sistema
  - Proporciona um guia para construção do sistema
  - Documenta as decisões tomadas.
- Os modelos são usados extensivamente na engenharia. Na construção civil, a construção de um prédio é precedida pela construção de modelos (desenhos)
- Da mesma forma linguagem UML fornece uma maneira para a modelagem de software através de desenhos

PRINCÍPIOS DA MODELAGEM

1. A escolha dos modelos a serem criados tem uma profunda influência sobre a maneira como um determinado problema é atacado e como uma boa solução é definida
2. Cada modelo poderá ser expresso em diferentes níveis de precisão
3. Os melhores modelos estão relacionados à realidade
4. Nenhum modelo é único e suficiente. Qualquer sistema não-trivial será melhor investigado por meio de um pequeno conjunto de modelos quase independentes.

VISÃO GERAL DA UML

- As linguagens fornecem um vocabulário e as regras para combinação de palavras desse vocabulário com a finalidade de comunicar
- Uma linguagem de modelagem é a linguagem que tem seu foco voltado para a representação conceitual e física de um sistema.
- O vocabulário e as regras de uma linguagem como a UML, indicam como criar e ler modelos bem-formados, mas não apontam quais modelos deverão ser criados, nem quando você deverá criá-los.

VISÃO GERAL DA UML

- A UML É UMA LINGUAGEM DESTINADA A:
  - VISUALIZAR…
  - ESPECIFICAR…
  - CONSTRUIR…
  - DOCUMENTAR…

  ...OS ARTEFATOS DE UM SISTEMA DE SOFTWARE.
- Artefato é um conjunto de informações utilizado ou produzido por um processo de desenvolvimento de software

UML – BLOCOS DE CONSTRUÇÃO

- O vocabulário de UML abrange três tipos de blocos de construção:
  - ITENS
  - RELACIONAMENTOS
  - DIAGRAMAS
- Os ITENS são as abstrações; Os RELACIONAMENTOS reúnem esses itens; Os DIAGRAMAS agrupam um conjunto de itens afins.

UML – ITENS

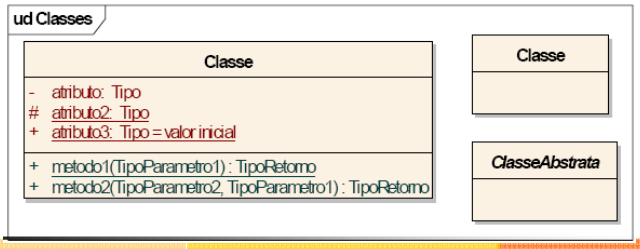
- Existem quatro tipos de itens na UML
  - Itens Estruturais – Substantivos utilizados no modelo; Partes mais estáticas; Representam elementos conceituais ou físicos
  - Itens Comportamentais – São as partes dinâmicas dos modelos UML. São os verbos utilizados no modelo, representando comportamentos no tempo e no espaço
  - Itens de Agrupamentos – São partes organizacionais dos modelos.
  - Itens Anotacionais – São as partes explicativas dos modelos UML

UML – ITENS ESTRUTURAIS

- Substantivos utilizados no modelo;
- São partes mais estáticas e representam elementos conceituais ou físicos
- A linguagem UML define os seguintes itens estruturais:
  - Classes
  - Interfaces
  - Casos de Uso
  - Colaborações
  - Classes Ativas
  - Componentes
  - Nós

UML – ITENS ESTRUTURAIS

- CLASSES – A seguir é mostrado como as classes são representadas em UML
- A UML sugere que o nome de classe comece com uma letra maiúscula e seja representado em negrito e no centro do compartimento superior.
- O compartimento do **meio contém os atributos e o** compartimento **inferior contém os métodos da classe**



UML – ITENS ESTRUTURAIS Classes

- ATRIBUTOS

  [visibilidade] [/] nome [: tipo] [[multiplicidade]]

  [= valor inicial][{propriedades}]

  - VISIBILIDADE
    - publico (+); protegido (#); privado (-); pacote (~)
  - / - Indica um atributo opcional, normalmente derivado de uma superclasse
  - TIPO
    - classe ou tipo primitivo de uma linguagem
  - MULTIPLICIDADE
    - Indica a quantidade de vezes que o atributo aparece. Pode ser representada da forma [inicio .. final] ou [total]
  - VALOR INICIAL
    - Valor padrão utilizado para o atributo
  - PROPRIEDADES
    - Características associadas aos atributos (“unique”; “ordered”; “readonly”; etc)

UML – ITENS ESTRUTURAIS Classes

- OPERAÇÕES (Métodos)

  [Visibilidade] nome (listaParametros) : tipoRetorno [{propriedades}]
- Visibilidade - publico (+); protegido (#); privado (-); pacote (~)
- Nome - Nome do método
- listaParametros
  - Parâmetros recebidos e/ou retornados pelo método. Podem ser escritos da seguinte forma:
  - **[direçãoParametro] nome : Tipo [[Multiplicidade]] [=valorPadrão]**
  - direçãoParâmetro
    - in – somente entrada; out – somente saída; inout – entrada e saída
  - valorPadrão – Valor padrão associado ao parâmetro
  - Tipo - classe ou tipo primitivo de uma linguagem
  - Multiplicidade - Indica a quantidade de vezes que o atributo aparece.
- Propriedades - podem representar “pré-condições”; “pós-condições”; “query”; etc.
- tipoRetorno - classe ou tipo primitivo de uma linguagem

UML – ITENS ESTRUTURAIS Classes - Abstratas

- Algumas classes na hierarquia são tão gerais que nenhum objeto será criado a partir delas. Neste caso a classe é dita ABSTRATA
- Uma classe abstrata não pode ser instanciada ou seja, não é possível criar objetos a partir da mesma
- A classe ABSTRATA é uma classe que está incompleta. Esta classe pode conter métodos abstratos que são aqueles métodos apenas declarados, mas que não foram implementados.
- Os métodos abstratos devem ser obrigatoriamente implementados nas subclasses

UML – ITENS ESTRUTURAIS
Classes - Abstratas

- O método abstrato contém apenas sua assinatura (nome, número e tipo dos seus parâmetros).
- Para a criação de classes e métodos abstratos deve ser utilizado o modificador de tipo que normalmente é a palavra “abstract”
- Classe CONCRETA é aquela a partir da qual objetos serão instanciados. Neste tipo de classe todos seus métodos devem ser, obrigatoriamente, definidos.

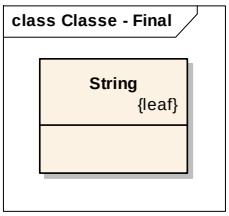
UML – ITENS ESTRUTURAIS
Classes - Modificadores

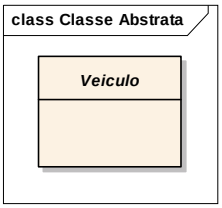
- Nas linguagens orientadas a objeto existem palavras reservadas chamadas Modificadores.
- Estas palavras modificam o comportamento de classes, atributos e métodos
- Por exemplo, na linguagem Java, temos os seguintes modificadores:

CLASSE	ATRIBUTO	MÉTODO
abstract		abstract
final	final	final
	static	static
	transient	
	volatile	
		synchronized
		native

UML – ITENS ESTRUTURAIS
Modificadores de Classe

- Abstract
  - Não é possível criar objetos de uma classe abstrata.
  - Além disso uma classe abstrata pode conter métodos abstratos que são aqueles que não possuem corpo, mas apenas sua assinatura.
- Final
  - Impede que uma determinada classe possa ser especializada, ou seja, impede a herança
  - Neste caso a classe não poderá ter nenhuma classe filha



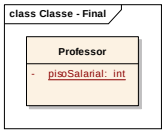


UML – ITENS ESTRUTURAIS
Modificadores de Atributo

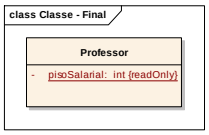
- Static
  - Conhecido como “atributo de classe”, este atributo está associado à classe e não a um objeto da classe.
  - Todos os objetos compartilham o mesmo atributo estático.
  - Uma mudança no atributo é percebida por todos os objetos da classe
- Final
  - Indica que o atributo é um valor constante e cujo valor não pode ser alterado.
  - O valor é associado ao atributo no momento em que o mesmo é inicializado e esta inicialização é feita no código
  - Exemplo: public static final PI = 3.141592653589793;
- Transient (Java)
  - Atributos transientes não pode ser serializados
  - Desta forma ao gravar os dados do objeto, no disco, por exemplo, este tipo de atributo será excluído do processo de serialização do objeto.
- Volatile (Java)
  - Em um código com vários threads (multithreaded) um atributo volátil conterá o mesmo valor em todos os threads.
  - Neste caso quando uma alteração é feita no atributo por um thread o mesmo será atualizado em todas as cópias, existentes em cada thread, daquele atributo existentes

UML – ITENS ESTRUTURAIS
Modificadores de Atributo

- Static



- Final

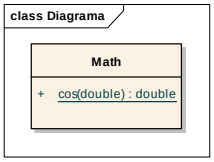


UML – ITENS ESTRUTURAIS
Modificadores de Método

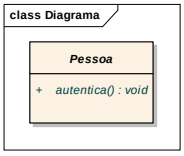
- Static
  - Conhecido como “método de classe”, este método está associado à classe
  - Desta forma é possível acessar o método sem a existência de um objeto da classe
  - Exemplo: p = dt * dU * dI * Math.cos(fi);
- Abstract
  - Neste caso o método somente a sua assinatura e não possui corpo
  - Esta presente em classes abstratas e em interfaces
- Final
  - Indica que o método não pode especializado em qualquer subclasse
  - Este modificador impede o polimorfismo
- Synchronized (Java)
  - Em um código com vários threads (multithreaded) garante que o método será executado de forma individual, por um único thread.
  - Quando um thread executar um método sincronizado, é necessário antes de mais nada, obter o lock do objeto. A partir deste ponto somente um thread poderá executar o código.
  - Ao final da execução o lock é liberado e o método poderá ser executado por outro thread
- Native (Java)
  - Indica que o método é implementando em uma outra linguagem, como C, C++ ou assembly
  - A JNI (Java Native Interface) é uma interface de programação que permite a execução de métodos nativos.

UML – ITENS ESTRUTURAIS
Modificadores de Método

- Static



- Abstract

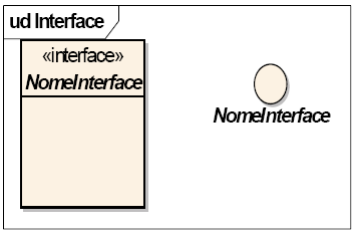


Praticando seus conceitos...

- A partir do modelo criado anteriormente, criar um diagrama de classes
- Utilizar uma ferramenta para a criação do diagrama
- Utilizar os conceitos discutidos até aqui:
  - Classes Abstratas
  - Métodos, Classes e Atributos modificados – Final, Static e Abstract

UML – ITENS ESTRUTURAIS
Interfaces

- Coleção de operações (métodos) que especificam os serviços de uma classe ou componente.
- A interface não contém a implementação das operações, mas apenas descreve quais são estas operações
- Através das interface é possível diminuir o acoplamento entre diferentes partes de um sistema
- Uma interface é realizada por uma ou mais classes

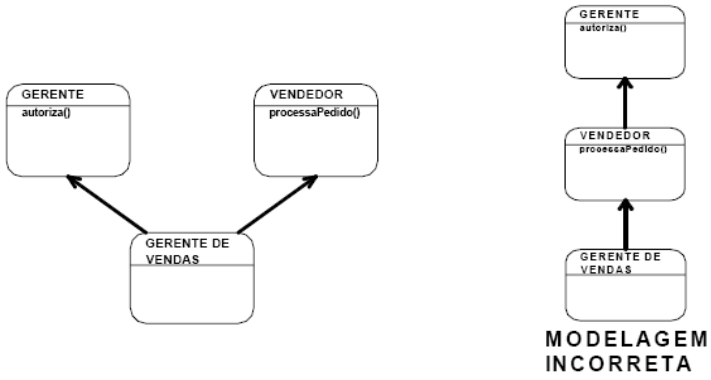


UML – ITENS ESTRUTURAIS
Interfaces

- O conceito de interface é um importante aliado no projeto de software
- Na linguagem Java, este conceito é utilizado para o modelamento da Herança Múltipla

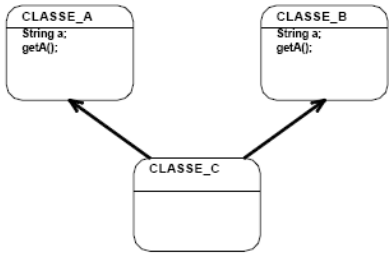
UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla

- Existem casos em que uma classe pode herdar o comportamento de mais de uma classe.
- Neste caso temos a herança múltipla, conforme mostrado abaixo



UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla (java)

- Como implementar a herança múltipla:



- Como implementar a herança múltipla:
- No exemplo acima, qual cópia do atributo **a a classe CLASSE_C** vai herdar? Qual método **getA() vai utilizar?**
- Java resolve este problema utilizando o conceito de “INTERFACES”

UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla (java)

- Um método possui duas partes: sua assinatura e sua Implementação
- Java não suporta a herança múltipla explicitamente, mas possui meios para que os efeitos da herança múltipla seja realizada de forma indireta utilizando o conceito de INTERFACES
- Através deste conceito uma classe pode herdar as assinaturas dos métodos, mas não a sua implementação.
- A implementação, deve por sua vez, ser definida na subclasse.
- Desta forma uma interface possui apenas um conjunto de métodos

UML – ITENS ESTRUTURAIS
Classes – Herança Múltipla (Java)

- Através da herança múltipla novos métodos,de diferentes classes, podem ser agregados a uma subclasse
- A herança através de interface não possibilita a reutilização do código, visto que o método herdado deve ser implementado para cada subclasse.
- ATRIBUTOS EM UMA INTERFACE
  - Em uma interface os atributos são implicitamente declarados como static e final.
- MÉTODOS EM UMA INTERFACE
  - Todos os métodos são abstratos, não sendo necessário a palavra “abstract”

UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla

- Na interface todos os métodos são abstratos e não possuem implementação apenas sua assinatura.
- A uma classe pode utilizar mais de uma interface em sua definição
- A interface representa um comportamento que a classe que a implementa irá obrigatoriamente possuir.

UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla (Java)

- Uma INTERFACE é definida através da palavra "interface" conforme mostrado a seguir:

  [public] interface B extends A
- Neste caso A deve ser outra interface.

  Exemplo – Definição da INTERFACE IGerente

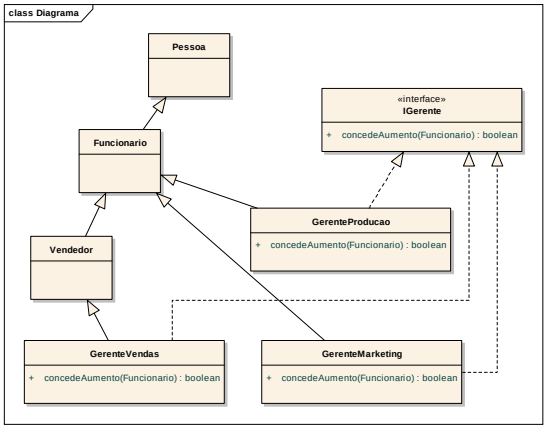
```
public interface IGerente{
   public void demitir(Empregado e);
   public boolean concedeAumento();
   public boolean contratar(Empregado e);
 }
```

  - O gerente de vendas, bem como qualquer outro tipo de gerente, realiza as operações demitir e contratar, porém cada um a seu modo
- A indicação da herança múltipla é feita da seguinte forma:

  [public] [modTipo] class B [extends A] implements C,[D,E,F..]

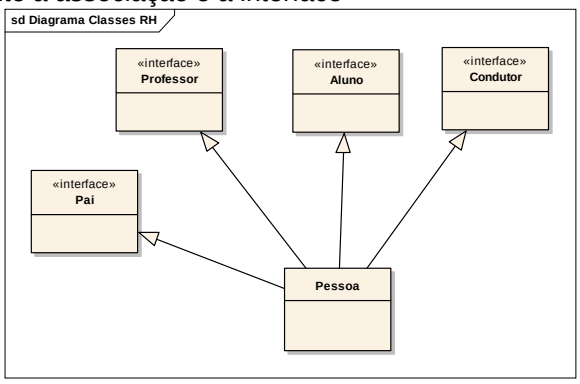
UML – ITENS ESTRUTURAIS
Interfaces – Comportamento Padrão

- Exemplo de Uso de Interface
- A interface garante que um conjunto de classes contenha um comportamento padrão (método), porém com as particularidades necessárias
- Cada tipo de Gerente, pode "concederAumento" porém segundo seus critérios particulares
- A interface garante que todos contenham o método



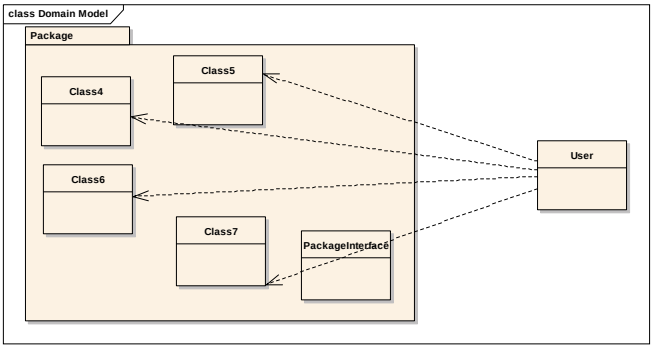
UML – ITENS ESTRUTURAIS
Interfaces – Comportamento Padrão

- Exemplo de Uso de Interface
- A classe Pessoa pode representar diversos papéis. Cada papel associa um conjunto de comportamentos (métodos)
- O que garante a associação é a interface



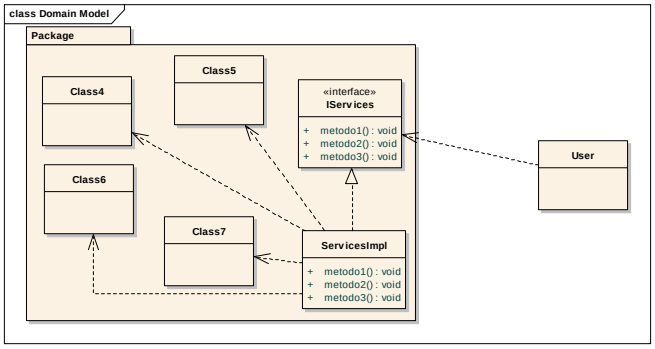
UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla

- Redução do Acoplamento
  - O diagrama abaixo mostra uma classe que utiliza outras classes
  - Neste caso existe um alto acoplamento entre as mesmas



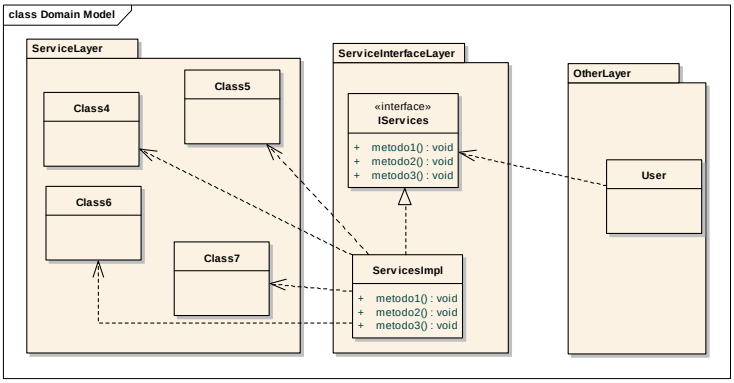
UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla

- Redução do Acoplamento
  - A partir do uso da uma Interface é possível reduzir o acoplamento
  - O único ponto de contato entre os módulos neste exemplo é a interface
  - Caso a interface seja mantida (assinatura dos métodos) os módulos são independentes entre si.



UML – ITENS ESTRUTURAIS
Interfaces – Herança Múltipla

- Redução do Acoplamento
  - A mesma informação, porém organizada de uma diferente forma



UML – ITENS ESTRUTURAIS
Casos de Uso

- Descrevem um conjunto de ações realizadas pelo sistema que proporciona resultados observáveis de valor para alguma entidade que está de fora deste sistema.
- Um caso de uso descreve um comportamento do sistema
- Um caso de uso é realizado por uma colaboração

UML – ITENS ESTRUTURAIS
Colaborações

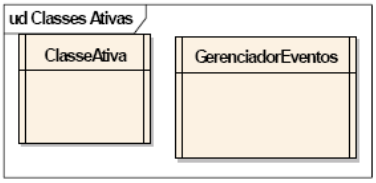
- É uma sociedade de classes,interfaces e outros elementos que trabalham em conjunto para fornecer algum comportamento cooperativo maior que a some de todas as suas partes.
- Uma determinada classe pode fazer parte de várias colaborações

UML – ITENS ESTRUTURAIS
Colaborações

- A colaboração possui dois aspectos:
  - Parte estrutural (classes; interfaces; componentes e nós)
  - Parte comportamental (como os elementos da parte estrutural se interagem)
- A colaboração não possui nenhum de seus elementos estruturais mas apenas faz referência a cada um deles, sendo portanto um grupo conceitual de itens.
- A colaboração pode ser utilizada para indicar a “realização” ou seja, a implementação do caso de uso
- Outro uso das colaboração é indicar a “realização” de uma operação

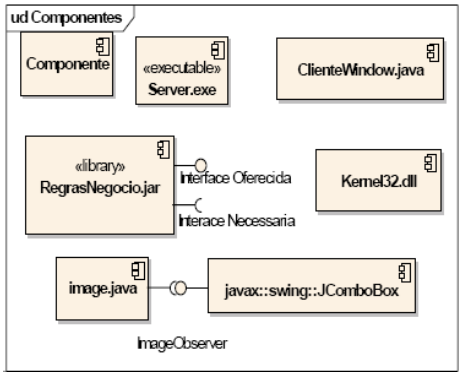
UML – ITENS ESTRUTURAIS
Classes Ativas

- Classes cujos objetos possuem um ou mais processos ou threads associados.
- Desta forma realizam alguma atividade de controle.
- São representadas conforme as classes, porém com duas linhas laterais
- Os objetos de classes ativas podem realizar operações concorrentes



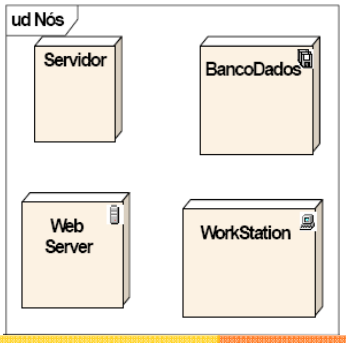
UML – ITENS ESTRUTURAIS
Componentes

- São partes físicas e substituíveis de um sistema que proporcionam a realização de um conjunto de interfaces
- Representam um pacote físico de classes; interfaces e colaborações



UML – ITENS ESTRUTURAIS
Nós

- Elemento físico que representa um recurso computacional, geralmente com alguma memória e capacidade de processamento.
- Um conjunto de componentes poderá estar em um nó.



Praticando seus conceitos...

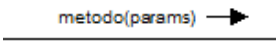
- Crie um conjunto de componentes relacionados ao sistema que você está trabalhando atualmente
- Relacione os casos de uso que você encontra no TQI-GP
- Considerando o conceito de Interface e sua representação mostre como duas classes podem se comunicar através de uma interface
- Da mesma forma, mostre como dois componentes podem se comunicar exclusivamente através de uma interface

UML – Itens Comportamentais

- São as partes dinâmicas dos modelos UML.
- São os verbos utilizados no modelo, representando comportamentos no tempo e no espaço
- Existem dois tipos de itens comportamentais
  - Interação
  - Máquina de estado
- Os itens comportamentais costumam estar ligados a vários elementos estruturais

UML – Itens Comportamentais Interação

- Um comportamento que abrange um conjunto de mensagens trocadas entre objetos, em determinado contexto, para a realização de um propósito
- As interações podem ser encontradas em qualquer parte em que os objetos estejam vinculados entre si.
- Existem no contexto de sistema ou subsistema; no contexto de uma operação e no contexto de uma classe.
- A modelagem dos aspectos dinâmicos é feita pela utilização de interações
- A interação é representada da seguinte forma:



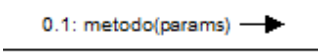
UML – Itens Comportamentais Interação

- Normalmente a linha cheia com a seta contém também o nome da mensagem que está sendo enviada.
- Vínculo
  - Um vínculo é uma conexão semântica entre objetos. Em geral um vínculo é uma instância da "associação".
- Sempre que existir uma associação de uma classe com outra poderá haver um vínculo entre elas e havendo o vínculo poderá haver a troca de mensagens.
- O vínculo especifica o caminho pelo qual o objeto pode enviar uma mensagem

UML – Itens Comportamentais Interação

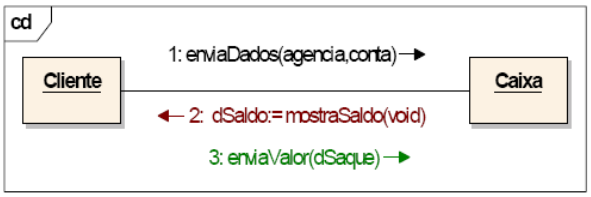
- Além da operação a interação pode conter outras informações:
- SEQÜENCIAMENTO
  - A fim de se obter a ordem em que as iterações acontecem em um contexto é possível associar as mesmas um número de seqüência.
  - O número de seqüência poderá ser da seguinte forma: i.j.k
  - No exemplo acima existem 3 níveis de chamadas
  - No caso da existência de múltiplos fluxos de controle (threads) pode ser utilizado o nome o thread que está controlando a interação:

    th2 : operation()



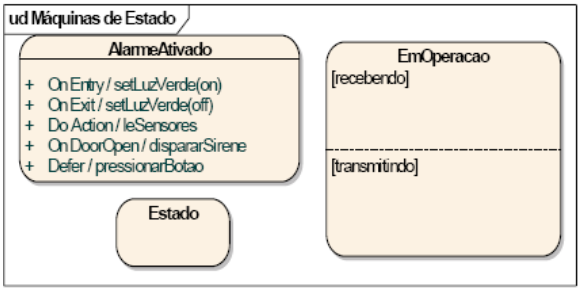
UML – Itens Comportamentais Interação

- Além disso a interação pode conter os argumentos que a operação utiliza e também valores de retorno.
  - 1.3.2 : p : find("Joao");



UML – Itens Comportamentais Máquinas de Estado

- Uma máquina de estado é um comportamento que especifica as seqüências de estados pelas quais objetos ou iterações passam durante sua existência em resposta a eventos, bem como suas respostas e estes eventos.
- As Máquinas de Estado podem ser compostas por um ou vários estados



UML – Itens Comportamentais Máquinas de Estado

- Conceitos Associados
  - EVENTO – Ocorrência significativa que tem uma localização no espaço e no tempo, capaz de provocar uma transição entre estados
  - ESTADO - condição ou situação da vida do objeto em que ele satisfaz alguma condição, realiza alguma atividade ou aguarda algum evento.
  - TRANSIÇÃO – Relacionamento entre dois estados

UML – Itens Comportamentais Máquinas de Estado

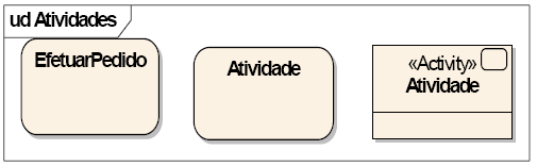
- As máquinas de estado podem representar ESTADOS DE AÇÃO ou ESTADOS DE ATIVIDADE.
  - ATIVIDADE – Execução **não atômica** em andamento em uma máquina de estados. Exemplo: EfetuarPedido; BuscarRegistros; etc.
  - AÇÃO – Execução **atômica** que resulta na alteração de estado ou no retorno de um valor.

UML – Itens Comportamentais Máquinas de Estado

- ESTADOS DE AÇÃO
  - Consiste de máquinas de estado que realizam ações.
  - Seu tempo de execução é considerado insignificante.
  - Um estado de ação não pode ser decomposto.
  - Exemplos:
    - Criação de um objeto;
    - Destruição de um objeto;
    - Enviar um sinal;
    - Receber um sinal

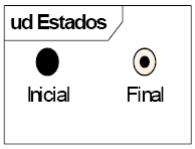
UML – Itens Comportamentais Máquinas de Estado

- ESTADOS DE ATIVIDADE
  - São estados que podem ser decompostos, sendo que suas atividades podem ser representadas por DIAGRAMAS DE ATIVIDADE.
  - Seu tempo de execução é não é considerado insignificante e além disso sua execução pode ser interrompida pois não são atômicos.



UML – Itens Comportamentais Máquinas de Estado

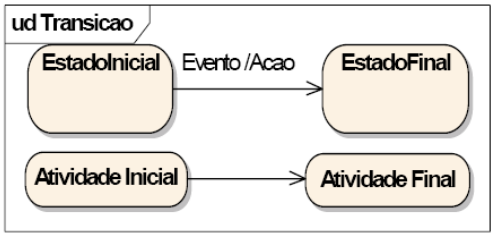
- Além disso existem dois estados especiais que possuem uma representação diferenciada.
- O ESTADO INICIAL e o ESTADO FINAL de uma máquina de estados.
- Um estado pode possuir:
  - Nome
  - Ações Entrada (“Entry") e Saída (“Exit")
  - Transições Internas
  - Atividade(“Do")
  - Evento Adiado (“Defer”)



UML – Itens Comportamentais Máquinas de Estado

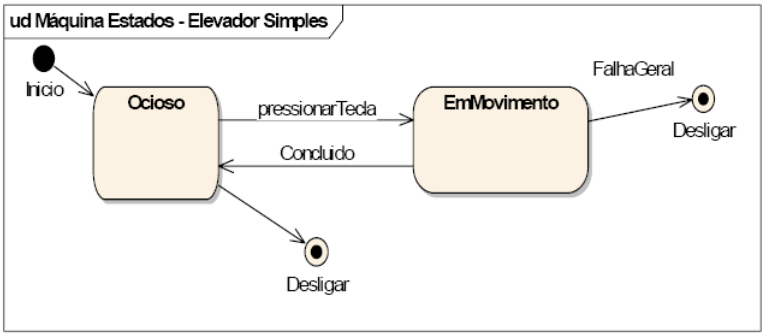
- TRANSIÇÃO
  - Relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará em um segundo estado quando um EVENTO especificado ocorrer.
  - Uma transição é a ligação entre dois estados
  - A transição pode conter a seguinte nomenclatura:

evento / ação



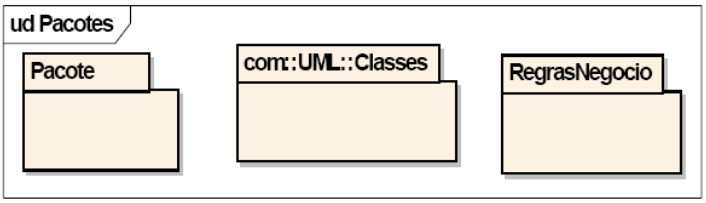
UML – Itens Comportamentais Máquinas de Estado

- O exemplo abaixo mostra uma máquina de estados utilizando todos os conceitos vistos anteriormente



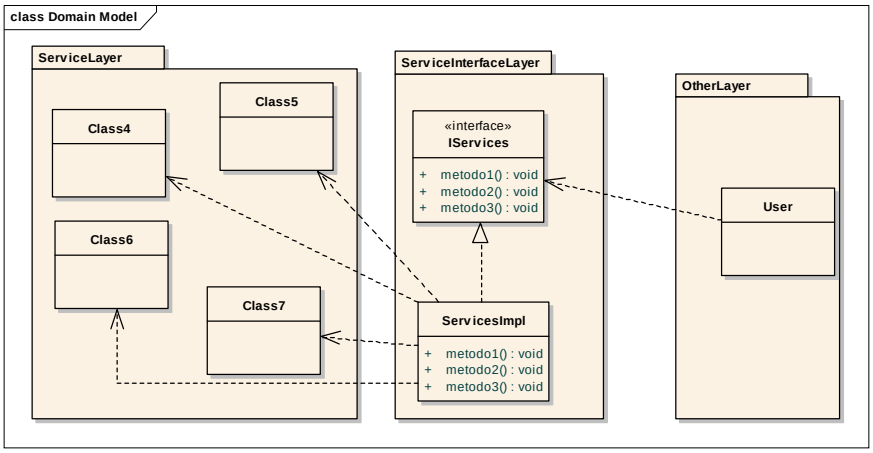
UML – Itens de Agrupamento

- São partes organizacionais dos modelos
- Existe um único tipo de item de agrupamento
  - Pacotes
- Os itens estruturais, comportamentais e até outros itens de agrupamento podem ser colocados em pacotes.
- Os pacotes são puramente conceituais e sua função é organizar os modelos UML



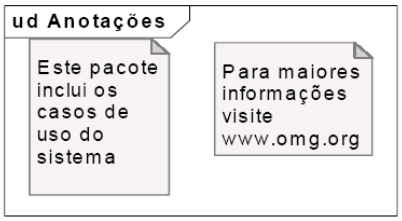
UML – Itens de Agrupamento Exemplo

- Exemplo de uso de pacotes na organização



UML – Itens de Anotacionais

- Existe um único tipo de item anotacional
  - Notas
- São comentários incluídos para descrever, esclarecer e fazer alguma observação sobre qualquer elemento do modelo.



Praticando seus conceitos...

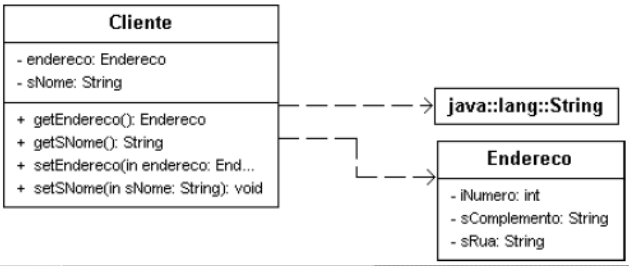
- Criar um diagrama de estados ligado à suas atividades diárias
- Criar uma máquina de estados que represente o estado “TelefoneEmUso”

UML – Relacionamentos

- Existem quatro tipos de relacionamentos na UML
  - Dependência
  - Associação
  - Generalização
  - Realização

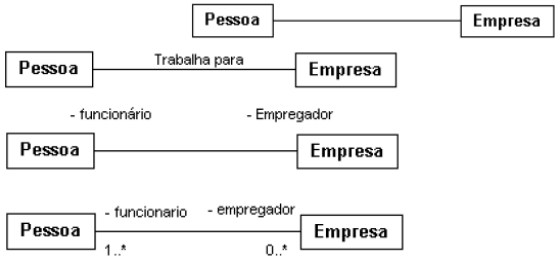
UML – Relacionamentos Dependência

- Relacionamento de utilização entre itens.
- Neste caso as modificações na especificação de um item (dependente) podem afetar o outro item que o utilize.
- Exemplo: A classe Cliente usa a classe String
- No contexto de classes a dependência mostra que uma classe utiliza a outra como argumento na assinatura de uma operação.



UML – Relacionamentos Associação

- Relacionamento estrutural que especifica um item conectado a outro item.
- Na associação as classes são pares umas das outras e realizam algum trabalho em conjunto, ou seja, uma colabora com a outra
- Exemplo: Salas são formadas por Paredes; Tubos estão inseridos em paredes



UML – Relacionamentos Associação

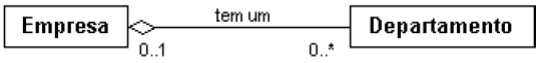
- Adornos básicos de uma associação:
  - Nome – O nome descreve a natureza do relacionamento
  - Papel – Indica um papel específico para a classe neste relacionamento
  - Multiplicidade – Indica a quantidade de objetos de uma classe que podem estar conectados à outra.
    - O valor padrão é muitos (0..*) mas podem haver valores como: um ou mais (1..*); exatamente 1 (1);
  - Navegação – Indica sentido de busca
  - Agregação / Composição – Consiste de um diamante aberto ou fechado que é colocado em uma das extremidades da associação

UML – Relacionamentos Associação

- Navegação
  - Adorno utilizado em uma associação
  - Considerando uma associação simples é possível navegar entre objetos de um tipo para outro tipo.
  - Em certos casos porém esta navegação deve ser limitada.
  - Exemplo: Na associação entre as classes Usuário e Senha, é possível encontrar a senha, a partir de um usuário mas o contrário não é possível. Desta forma a navegação é colocada apenas em uma direção, neste caso do usuário para a senha

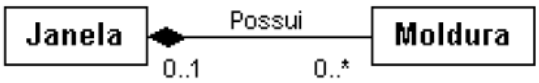
UML – Relacionamentos Associação

- AGREGAÇÃO
  - Neste tipo de associação, as classes estão em um mesmo nível, porém neste caso deve ser feito a modelamento “todo/parte”.
  - O item maior (o todo) é formado por itens menores (as partes).
  - Este é um relacionamento do tipo: A “tem um” B.
  - Para representar o todo é colocado um diamante aberto na extremidade do todo.



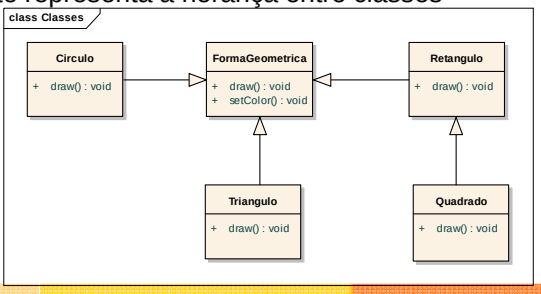
UML – Relacionamentos Associação

- COMPOSIÇÃO
  - A composição é uma forma de AGREGAÇÃO, com propriedade bem definida e tempo de vida coincidente como parte do todo.
  - Neste tipo de associação o “todo” é responsável pela criação e destruição das “partes”.
  - Para representar o todo é colocado um diamante fechado na extremidade do todo.



UML – Relacionamentos Generalização

- Relacionamento de especialização/realização nos quais os objetos dos elementos especializados (filhos) são substituíveis por objetos do elemento generalizado (pais).
- Filhos compartilham estrutura e o comportamento dos pais
- Pode-se dizer o filho "**é um tipo do**" pai ou seja: Um retângulo é um tipo de forma geométrica
- Este relacionamento representa a herança entre classes



UML – Relacionamentos Generalização

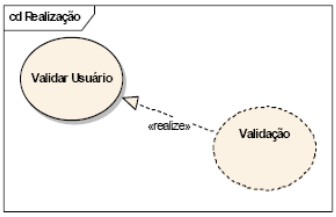
- A generalização significa que os objetos da classe filha podem ser utilizados em qualquer local em que a classe pai ocorra, mas não vice-versa
  - Um método que recebe como parâmetro uma FiguraGeometrica poderá receber qualquer objeto das classes: Circulo; Triangulo ou Quadrado
- Caso a classe filha possua uma operação que tenha a mesma assinatura de uma operação da classe pai, esta operação prevalecerá
  - No exemplo anterior ao chamar o método draw() na classe Quadrado, este será executado a não o método draw() da classe Retangulo
  - Por sua vez, o método setColor() está disponível para a classe Quadrado e poderá ser utilizado por um objeto desta classe
  - O método draw() é um exemplo de Polimorfismo
- Caso uma classe filha possua mais de uma classe pai, diz-se que é um caso de herança múltipla
- A classe filha que não possuem outras classes filhas é chamada de folha

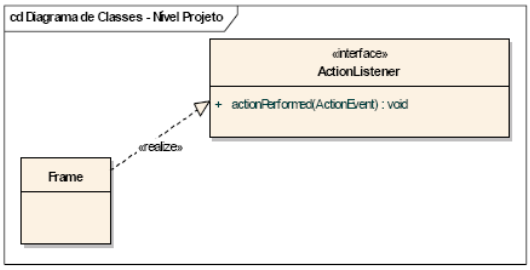
UML – Relacionamentos Realização

- Relacionamento semântico entre classificadores, onde um classificador especifica um contrato que outro classificador garante executar.
- Classificador é um item que pode apresentar instâncias e que possui características estruturais e comportamentais.
  - Incluem classes; Interfaces; Tipos de Dados; Sinais; Componentes; Nós; Casos de Uso e Subsistemas (pacotes)
- A realização é utilizada no contexto das interfaces e colaborações

UML – Relacionamentos Realização

- Exemplos





Praticando seus conceitos...

- Crie um modelo de Tabelas (semelhante a um DER), porém utilizando a notação da linguagem UML
- Criar um diagrama de classes que contenhas as seguintes classes:
  - Cada item da Linguagem UML
  - Cada relacionamento da Linguagem UML
  - Classe Diagrama

UML – Regras

- Existem regras para a combinação dos blocos de construção.
  - Nomes – Quais nomes podem ser atribuídos
  - Escopo – Contexto que determina significado específico para cada nome
  - Visibilidade – Como os nomes podem ser vistos e utilizados por outros
  - Integridade – Como os itens se relacionam entre si de forma adequada e consistente
  - Execução – O que significa executar ou simular um modelo dinâmico

UML – Regras da Linguagem

- Como toda linguagem existem regras associadas à linguagem UML
- Os blocos de construção da linguagem (itens; relacionamentos e diagramas) não podem ser combinados de forma aleatória
- As regras da linguagem especificam o que será um modelo bem formado
- A UML possui regras semânticas para:
  - Nomes – Quais nomes podem ser atribuídos
  - Escopo – Contexto que determina significado específico para cada nome
  - Visibilidade – Como os nomes podem ser vistos e utilizados por outros
  - Integridade – Como os itens se relacionam entre si de forma adequada e consistente
  - Execução – O que significa executar ou simular um modelo dinâmico

UML – Regras da Linguagem Modelos

- Os modelos podem se apresentar da seguinte forma:
  - Modelos bem formados(Consistentes) – São modelos corretos e escritos conforme todas as regras da UML
  - Parciais – Certos elementos ficam ocultos para simplificar a visão do modelo
  - Incompletos – Certos elementos podem ser omitidos
  - Inconsistentes – A integridade do modelo não é garantida
- Desta forma a linguagem permite não apenas a presença de modelos “bem formados” mas aceita simplificações ou omissões

UML – Mecanismos da Linguagem

- Mecanismos básicos da linguagem:
  - ESPECIFICAÇÕES – Por trás de cada bloco existem uma série de especificações que permitem sua construção de forma incremental, além de possibilitar sua visão de diferentes formas.
  - ADORNOS – Permitem acrescentar outros detalhes a um um bloco de construção. Exemplo:

    Classe - Visibilidade; Classe Abstrata

    Associação - Multiplicidade; Papel
  - DIVISÕES COMUNS – Blocos podem ser divididos em: Classes e Objetos
  - MECANISMOS DE EXTENSÃO – Permitem uma ampliação controlada da linguagem

UML – Mecanismos da Linguagem Extensão

- Entre os mecanismos de extensão na UML temos:
  - ESTEREÓTIPOS – Ampliam o vocabulário da UML, permitindo a criação de novos blocos. Ex.:<<Interface>>; <<exception>>
  - VALORES ATRIBUIDOS – Estende as propriedades dos blocos.

    Ex: {versao = 1.3}{autor = FOS}
  - RESTRIÇÕES – Amplia a semântica permitindo acrescentar regras ou modificar as já existentes.

    Ex: {ordered}; {velocidade > 10 Mbps}

UML – Diagramas

- Um diagrama é a apresentação gráfica de um conjunto de elementos, onde os vértices são ITENS e os arcos RELACIONAMENTOS
- UML 2.0 possui os seguintes diagramas:
  - Diagrama de Classes (Class Diagram)
  - Diagrama de Objetos (Object Diagram)
  - Diagrama de Componente (Component Diagram)
  - Diagrama de Implantação (Deployment Diagram)
  - Diagrama de Composição (Composite Struture Diagram)
  - Diagrama de Pacotes (Package Diagram)
  - Diagrama de Caso de Uso (Use Case)
  - Diagrama de Seqüência (Sequence Diagram)
  - Diagrama de Comunicação (Communication Diagram)
  - Diagrama de Atividade (Activity Diagram)
  - Diagrama de Estados (State Machine Diagram)
  - Diagrama de Interação (Interaction Overview Diagram)
  - Diagrama de Tempo (Timing Diagram)

UML – Diagramas

- Os diagramas da UML podem ser divididos em dois grandes grupos: DIAGRAMAS ESTRUTURAIS e DIAGRAMAS COMPORTAMENTAIS
- DIAGRAMA ESTRUTURAIS
  - Permitem modelar os aspectos estáticos de um sistema.
- DIAGRAMA COMPORTAMENTAIS
  - Permitem modelar os aspectos dinâmicos de um sistema.
  - Os aspectos dinâmicos são as partes do sistema que sofrem alteração durante a utilização do sistema

UML – Diagramas Estruturais

- Diagrama de Classes (Class Diagram)
- Diagrama de Objetos (Object Diagram)
- Diagrama de Componente (Component Diagram)
- Diagrama de Implantação (Deployment Diagram)
- Diagrama de Estrutura Composta (Composite Struture Diagram)
- Diagrama de Pacotes (Package Diagram)

UML – Diagramas Comportamentais

- Diagrama de Caso de Uso (Use Case)
- Diagrama de Seqüência (Sequence Diagram)
- Diagrama de Comunicação (Communication Diagram)
- Diagrama de Atividade (Activity Diagram)
- Diagrama de Estados (State Machine Diagram)
- Diagrama de Geral Interação (Interaction Overview Diagram)
- Diagrama de Tempo (Timing Diagram)

UML – Diagramas Comportamentais Caso de Uso

- Este diagrama representa um conjunto de casos de uso, atores e seus relacionamento.
- Permitem visualizar o comportamento de um sistema, subsistema ou classe a fim de que desenvolvedores e usuários possam se comunicar.
- Usos
  - Modelagem do Contexto do Sistema e Modelagem dos Requisitos de um Sistema
- Modelagem do Contexto do Sistema
  - Mostra o sistema considerando os atores que se encontram de fora do mesmo e sua ligação com o sistema.
  - Este diagrama é semelhante a um DFD de contexto.
- Modelagem dos Requisitos do Sistema
  - Neste caso o diagrama irá representar um requisito do projeto.
  - A preocupação não é "como" o sistema faz mas "o que" o sistema faz. Este tipo de diagrama está ligado à fase de ANÁLISE de cada uma das iterações.

UML – Diagramas Comportamentais Caso de Uso

- Normalmente os casos de uso possuem uma descrição que contém uma combinação de nome/verbo
  - Exemplo: Criar Conta; Realizar Venda; Efetuar Pedido
- Os atores são aqueles que interagem com o caso de uso
  - Exemplos: Pessoas; Equipamentos; Outros Sistemas;
- Casos de uso definem o escopo do sistema, facilitando o entendimento da complexidade e tamanho do mesmo.
- A "soma" dos casos de uso deve ser igual ao sistema como um todo. Desta forma o que não é mostrado como caso de uso, não é parte do sistema
- O caso de uso descreve "o que" um sistema (ou subsistema) faz, mas não descreve "como" isto é feito

UML – Diagramas Comportamentais Caso de Uso

- Os casos de uso podem ser organizados segundo suas características:
  - Casos de Alto Risco
  - Casos básicos da Arquitetura
  - Casos que utilizam amplas funcionalidades do sistema
  - Casos de uso que necessitam de pesquisa ou que utilizam novas tecnologias
  - Casos de Uso Acessórios
- A partir da classificação dos casos de uso as iterações devem ser planejadas, com o foco nos casos de uso de maior prioridade, conforme a classificação realizada

UML – Diagramas Comportamentais Caso de Uso

- Os casos de uso são utilizados para a comunicação entre o desenvolvedor e os usuários, devido a sua simplicidade
- Os caso de uso são a base do desenvolvimento, pois podem ser utilizados para o planejamento de todas as fases do desenvolvimento e de cada uma das iterações.
- Os casos de uso podem ser utilizados como base a criação de testes do sistema
- Pode-se também utilizá-los como base para a criação da documentação para o sistema.

UML – Diagramas Comportamentais Caso de Uso

- Complexidade
  - Um caso de uso não deve ser complexo
  - Basicamente o caso de uso deve: “Satisfazer o objetivo do Ator”
  - Esta regra é valida principalmente nas etapas iniciais
  - Posteriormente, o caso de uso de pode ser decomposto e detalhado nas etapas seguintes

UML – Diagramas Comportamentais Caso de Uso

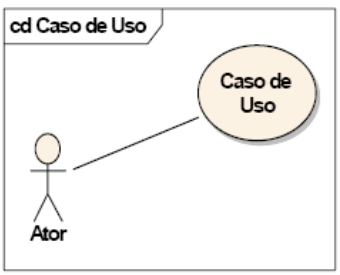
- Complexidade
  - Considere uma série de iterações entre o usuário e um sistema para saques bancários:
    - Inserir Cartão
    - Entrar com a Senha
    - Selecionar o Valor a ser sacado
    - Confirmar o valor a ser sacado
    - Remover o Cartão
  - Nas etapas iniciais não é necessário um caso de uso com tantos detalhes.
  - A complexidade será tratada nas fases seguintes, inclusive,podendo ser utilizados outros diagramas
  - O ideal é que o diagrama acima tenha apenas um caso de uso: “Realizar Saque”

UML – Diagramas Comportamentais
Caso de Uso - Relacionamentos

- Os seguintes relacionamentos são utilizados no diagrama de casos de uso
  - Associação
  - Generalização
  - Extensão
  - Inclusão

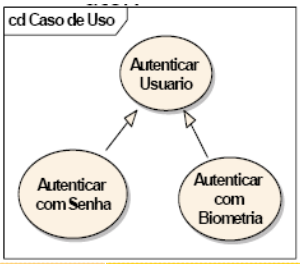
UML – Diagramas Comportamentais
Caso de Uso - Relacionamentos

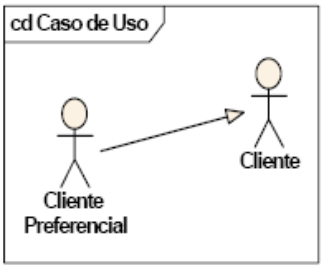
- Associação
  - Representa a ligação entre um ator e um caso de uso
  - O ator pode ser um outro sistema, um usuário, um dispositivo de hardware; etc.



UML – Diagramas Comportamentais
Caso de Uso - Relacionamentos

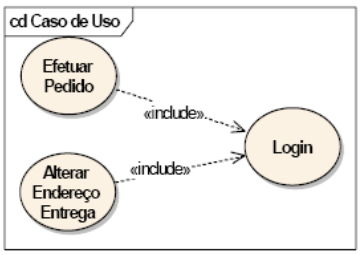
- Generalização
  - A generalização indica que o caso de uso filho herda o comportamento do caso de uso pai
  - Porém o caso de uso filho contém terá comportamentos que sobrescrevem aqueles do pai ou mesmo comportamentos particulares.
  - A generalização pode ser aplicada tanto ao caso de uso como ao ator.

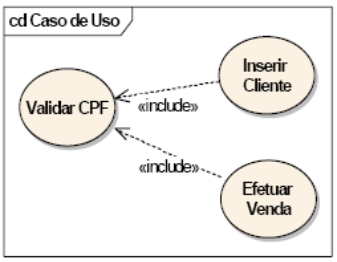




UML – Diagramas Comportamentais
Caso de Uso - Relacionamentos

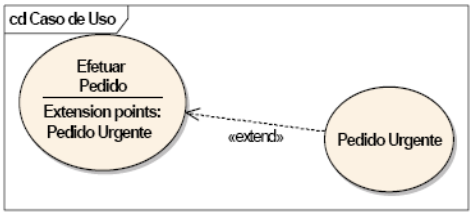
- Inclusão
  - Na relação de inclusão um caso de uso, inclui em seu comportamento, o comportamento de outro caso de uso
  - A inclusão permite que o caso de uso se complemente adicionando o comportamento do caso de uso incluído
  - O caso de uso depende do caso de uso incluído para efetuar suas operações





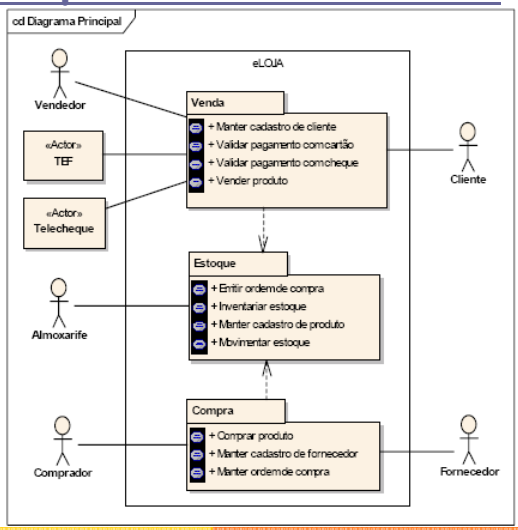
UML – Diagramas Comportamentais
Caso de Uso - Relacionamentos

- Extensão
  - A extensão de um caso de uso é utilizada para modelar um comportamento opcional
  - Desta forma separa-se o comportamento obrigatório do comportamento opcional
  - Os pontos em que o caso de uso pode ser estendido são conhecidos como “pontos de extensão” e devem ser bem definidos
  - Através da extensão é possível adicionar diferentes comportamentos a um caso de uso



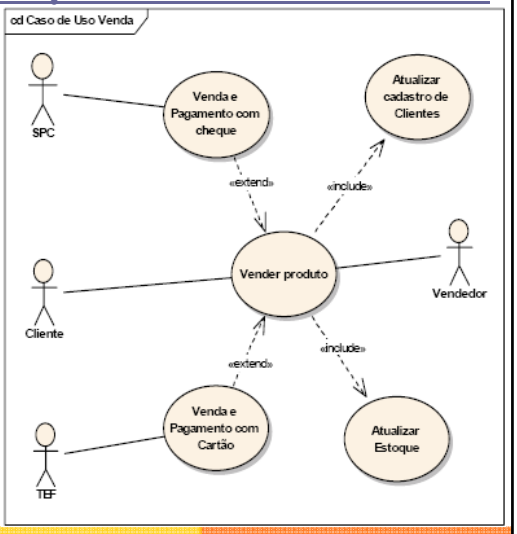
UML – Diagramas Comportamentais
Caso de Uso - Exemplo

- Caso de uso que mostra a visão geral de um sistema com os seus principais casos de uso e atores envolvidos
- Os casos de usos estão em pacotes.
- Alguns atores como TEF, SPC e fornecedor na realidade representam outros sistemas



UML – Diagramas Comportamentais
Caso de Uso - Exemplo

- Caso de uso que mostra uma parte do sistema anterior em maiores detalhes
- O caso de uso principal – "Vender Produto" foi um pouco detalhado indicando diferentes operações de venda que podem ser realizadas e ainda a possível necessidade de atualizar o cadastro de cliente no momento da venda.



Praticando seus conceitos...

- Considerando o sistema que você está trabalhando atualmente criar um diagrama de caso de uso, utilizando os seguintes conceitos:
  - Associação; Generalização; Inclusão e Extensão
- Considere um sistema na Web que será responsável por gerenciar contatos.
  - Além de cadastrar os contatos é possível; consultar os contatos; alterar suas informações; imprimir seus dados e finalmente enviar e-mail para um determinado contato

UML – Diagramas Estruturais
Classes

- Mostra um conjunto de classes, interfaces e colaborações bem como seus relacionamentos
- O diagrama de classes representa aspectos estruturais de um software
- No uso da Orientação a Objetos em última análise o objetivo das etapas concepção e Elaboração é estabelecer quais as classes serão criadas, quais seus atributos e os seus métodos, além dos relacionamentos entre estas várias classes.

UML – Diagramas Estruturais
Classes

- A medida que um software é modelado as classes necessárias bem como suas características como atributos e métodos vão sendo detalhadas.
- Existem classes que estão ligadas ao **domínio do problema. Neste caso as classes representam objetos** existentes no mundo real e ligados a um software em questão.

  Exemplos: Pedido; Cliente; Produto; Empregado; Paciente; Curso; etc.
- Existem classe que estão ligadas ao **domínio da solução e** que existem para representar objetos existentes do ponto de vista lógico para a construção de um software.

  Exemplos: Janela; Fila; Lista; Pilha; Aplicacao; Menu; Form; Window; Vector; Conexao; HttpServlet; etc.

UML – Diagramas Estruturais Classes - Usos

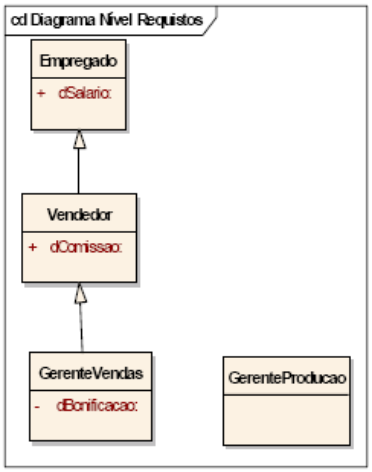
- O diagrama de classes pode ser representado em vários níveis de detalhamento
- Diagrama de Classes em nível de Modelamento do Negócio e Especificação Requisitos
  - Neste caso o diagrama contém poucos detalhes sobre cada classe, visto que as características do sistema, seu escopo e requisitos ainda estão sendo especificados
- Diagrama de Classes em nível de Análise
  - Contém mais detalhes sobre as classes como seus atributos e alguns métodos
  - Em geral representa apenas classes ligadas ao domínio do problema
- Diagrama de Classes em Nível de Projeto
  - Contém todos os detalhes referentes aos atributos e ao métodos como seu tipo, já mapeado em alguma linguagem de programação, os parâmetros de cada método e seu tipos de retorno
  - Contém classes ligadas tanto ao domínio do problema quanto ao domínio da solução.

UML – Diagramas Estruturais Classes - Usos

- USOS: Modelamento do Vocabulário; Modelamento de colaborações; Modelamento lógico do banco de dados
  - **Modelamento do Vocabulário**
    - O diagrama de classes pode ser utilizado para a modelagem do vocabulário do sistema que consiste nas classes, seus atributos e operações
  - **Modelamento de Colaborações**
    - Um colaboração é um conjunto de classes que realizam determinados papéis. Através do diagrama de classes é possível mostrar a sua parte estrutural
  - **Modelamento Lógico do Banco de Dados**
    - Os diagramas de classes da UML são um superconjunto dos diagramas de entidade-relacionamento (E-R). Os diagramas de classes vão um pouco além, permitindo ainda a modelagem de comportamentos, que poderão ser tornar procedimentos armazenados (stored procedures)
- Um mesmo aspecto do sistema pode ser representado por diferentes diagramas de classe.
- Exemplo: um diagrama pode privilegiar apenas os relacionamentos de herança, outro as dependências entre as classe e outro ainda as associações entre as mesmas

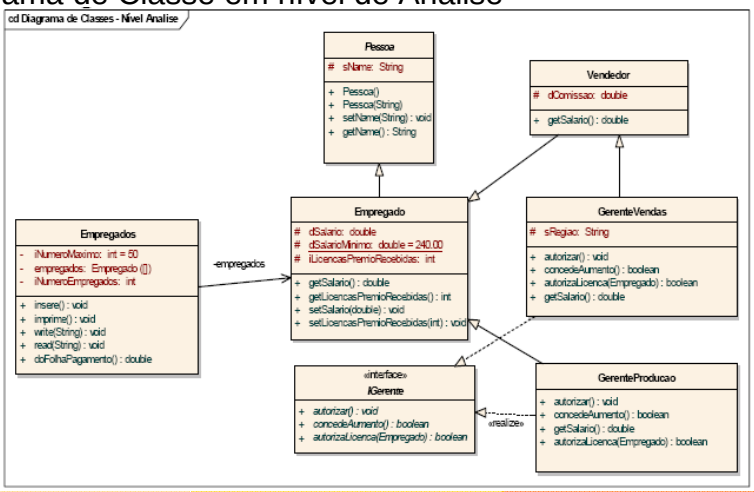
UML – Diagramas Estruturais Classes - Exemplos

- Diagrama de Classe em nível de Especificação de Requisitos



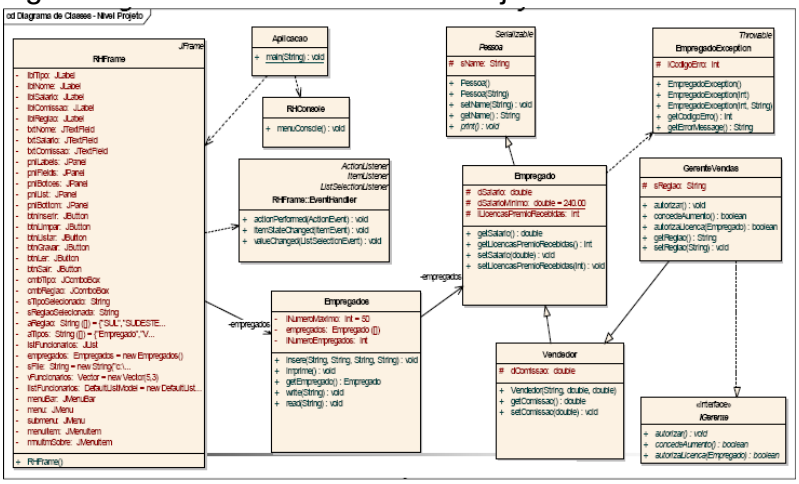
UML – Diagramas Estruturais Classes - Exemplos

- Diagrama de Classe em nível de Análise



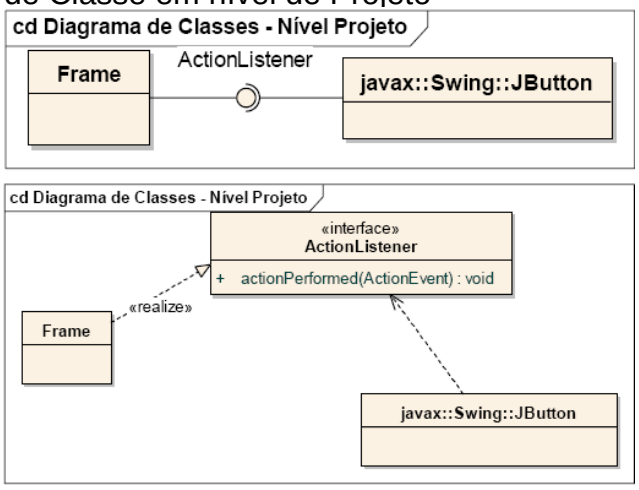
UML – Diagramas Estruturais Classes - Exemplos

- Diagrama de Classe em nível de Projeto



UML – Diagramas Estruturais Classes - Exemplos

- Diagrama de Classe em nível de Projeto

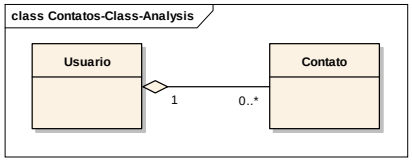


Praticando seus conceitos...

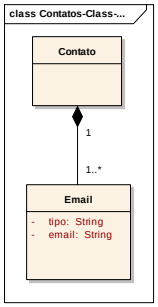
- Considere um sistema na Web que será responsável por gerenciar contatos, conforme caso de uso anterior
  - Além de cadastrar os contatos é possível; consultar os contatos; alterar suas informações; imprimir seus dados e finalmente enviar e-mail para um determinado contato
- Informações adicionais
  - Um usuário pode possuir vários contatos e o sistema deverá manter os dados de cada usuário individualmente
  - Um contato pode possui até 4 diferentes endereços de e-mail e para cada e-mail está associado um tipo (comercial; particular; prioritário; final de semana; etc.)
  - As informações associadas ao contato são as seguintes: Nome; Telefone Principal; Email
- Construir os diagramas de classe em nível de análise
- Construir um diagrama de classe que representa a modelagem de dados

Praticando seus conceitos...

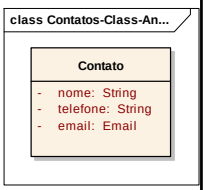
- Informações adicionais x Representação em UML
  1. Um usuário pode possuir vários contatos e o sistema deverá manter os dados de cada usuário individualmente
  2. Um contato pode possui vários endereços de e-mail e para cada e-mail está associado um tipo (comercial; particular; prioritário; final de semana; etc.)
  3. As informações associadas ao contato são as seguintes: Nome; Telefone Principal; Email



Regra 1



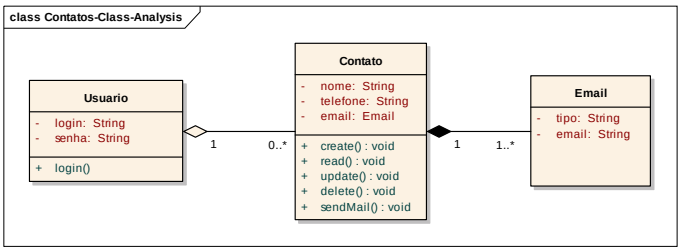
Regra 2



Regra 3

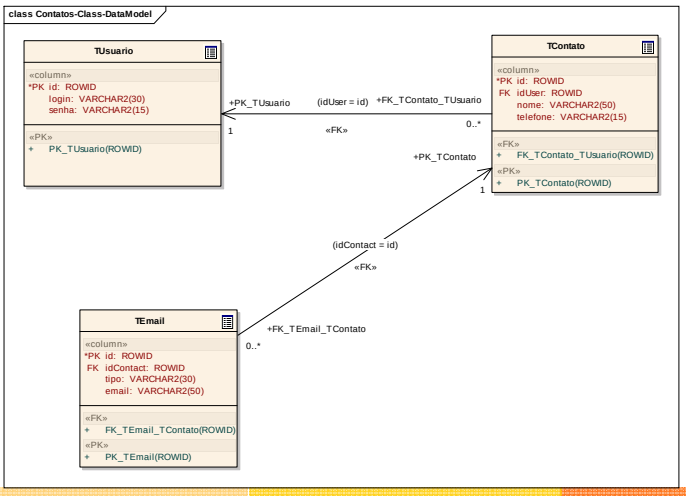
Praticando seus conceitos...

- Diagramas de classe em nível de análise
  1. Um usuário pode possuir vários contatos e o sistema deverá manter os dados de cada usuário individualmente
  2. Um contato pode possui vários endereços de e-mail e para cada e-mail está associado um tipo (comercial; particular; prioritário; final de semana; etc.)
  3. As informações associadas ao contato são as seguintes: Nome; Telefone Principal; Email



Praticando seus conceitos...

- Diagrama de classe que representa a modelagem de dados



UML – Diagramas Estruturais Objetos

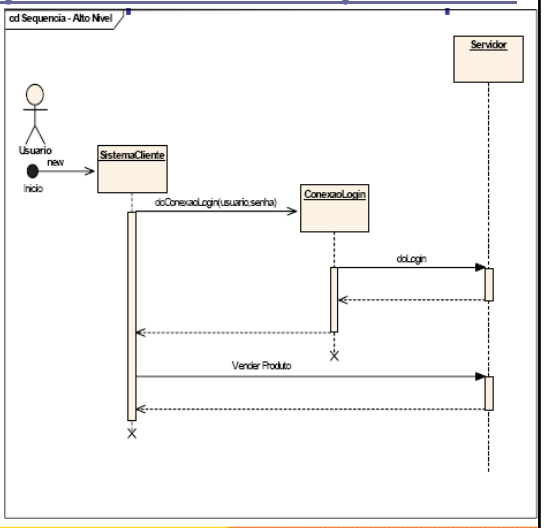
- Mostra um conjunto de objetos e seus relacionamentos em um ponto do tempo.
- Representa uma visão estática de um momento da execução do sistema
- USOS: Modelagem de Estruturas de Objetos
- Modelagem de Estruturas de Objetos
  - Mostra os relacionamentos estáticos em um determinado momento, permitindo a criação de um protótipo permitindo a exposição de conjunto de objetos, onde seu conhecimento é interessante.
- Modelagem de Fluxos de Controle Por Ordenação Temporal
  - Neste caso é possível mostrar a interação entre objetos que podem estar presentes em um sistema, subsistema, operação ou classe.
  - Além disso é possível também mostrar a interação entre objetos e papéis (atores) que participam em um caso de uso ou colaboração

UML – Diagramas Comportamentais Diagrama de Seqüência

- Este diagrama mostra as interações entre um conjunto de objetos e seus relacionamentos, incluindo as mensagens que serão trocadas entre os mesmos.
- Consiste de uma tabela que mostra objetos distribuídos no eixo X e mensagens em ordem crescente de tempo no eixo Y.
- Os **Diagramas de Seqüência são** conhecidos também como **Diagramas de Interação e a partir de um diagrama de** seqüência é possível obter o diagrama de comunicação a ele relacionado

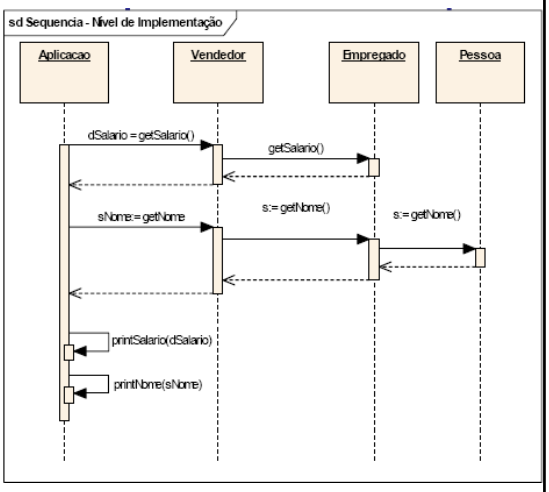
UML – Diagramas Comportamentais
Diagrama de Seqüência - Exemplo

- Diagrama de Seqüência em nível de modelamento de negócio e especificação de requisitos
- Mostra um fluxo de trabalho, indicando as mensagens trocadas no tempo.
- Neste caso as mensagens ainda estão em um alto nível de abstração



UML – Diagramas Comportamentais
Diagrama de Seqüência - Exemplo

- Diagrama de Seqüência em nível implementação
- Realiza o modelamento de um método (ou operação)
- Neste caso as mensagens ainda estão em um baixo nível de abstração, sendo que o diagrama está muito próximo do código-fonte

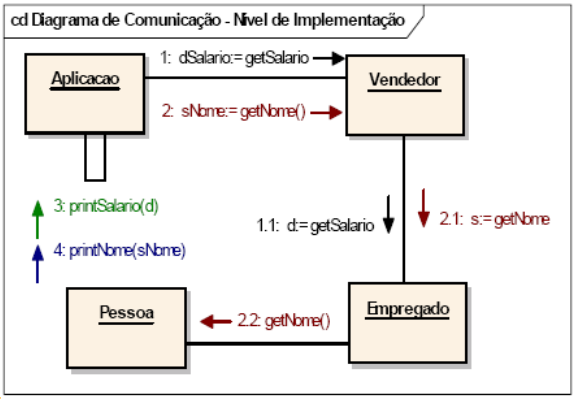


UML – Diagramas Comportamentais
Diagrama de Comunicação

- Este diagrama mostra as interações entre um conjunto de objetos e seus relacionamentos, incluindo as mensagens que serão trocadas entre os mesmos, considerando a ordem seqüencial que estas mensagens ocorrem entre os vários objetos.
- Os **Diagramas de Comunicação são conhecidos** também como **Diagramas de Interação e a** partir de um diagrama de comunicação é possível obter o diagrama de seqüência a ele relacionado
- USOS: Modelagem de Fluxos de Controle Por Organização

UML – Diagramas Comportamentais
Diagrama de Comunicação - Exemplo

- Diagrama de comunicação em nível de implementação
- Este diagrama é análogo ao diagrama de seqüência apresentado anteriormente



UML – Diagramas Comportamentais Diagrama de Atividades

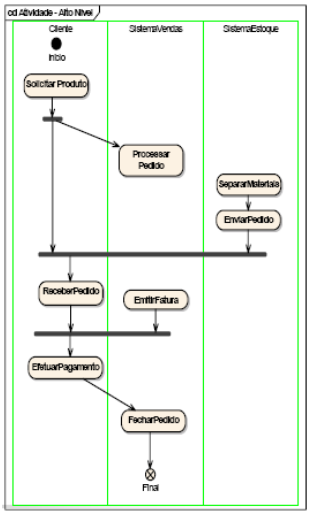
- Este diagrama mostra o fluxo de uma atividade para outra.
- Uma atividade é uma execução em andamento em uma máquina de estados.
- Um diagrama de atividades pode ser decomposto em outro diagrama de atividades e além disso as atividades podem conter operações de alto nível.
- A utilização das “raias de natação” são úteis para mostrar processos de negócio.
- USOS: Modelagem de Fluxo de Trabalho (nível especificação); Modelagem de uma Operação (nível projeto);

UML – Diagramas Comportamentais Diagrama de Atividades

- Modelagem do Fluxo de Trabalho
  - Através do diagrama é possível modelar os processo de negócios e regras, considerando o relacionamento entre vários sistemas e atores
  - É possível especificar e detalhar regras de negócios em sistemas, que muitas das vezes podem ser complexas, principalmente em sistemas de missão crítica e corporativos
  - Pode ser utilizado para modelar os fluxos de um caso de uso
- Modelagem de uma Operação
  - Neste caso o diagrama de atividades é semelhante a um fluxograma das ações de uma operação.
  - A vantagem neste caso é que todos os elementos apresentados neste diagramas são semanticamente relacionados com outros diagramas.

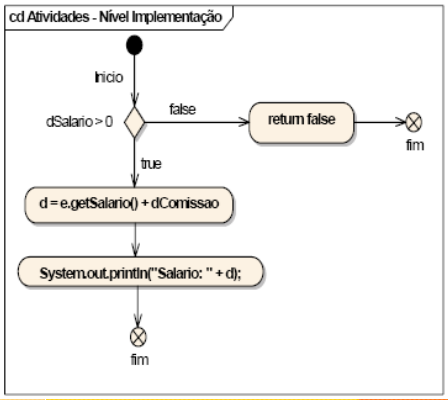
UML – Diagramas Comportamentais
Diagrama de Atividades - Exemplo

- Diagrama de Atividades em alto nível
  - Pode ser utilizado para detalhar aspectos de um caso de uso
  - Desta forma o diagrama auxiliará no entendimento do caso de uso e nas atividades que serão realizadas



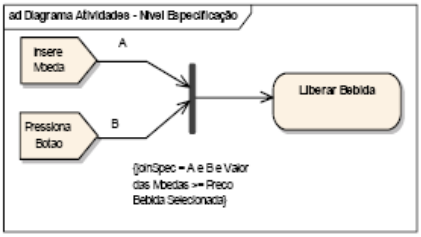
UML – Diagramas Comportamentais
Diagrama de Atividades - Exemplo

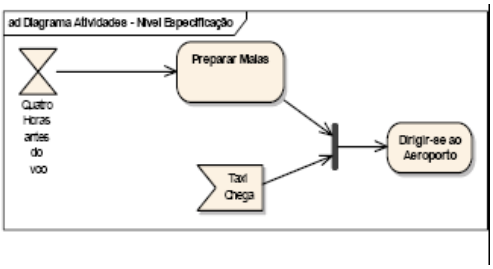
- Diagrama Atividades em nível de implementação
  - Neste caso a atividade representa a chamada de métodos
  - Pode ser utilizado para representar aspectos do código-fonte



UML – Diagramas Comportamentais
Diagrama de Atividades - Exemplo

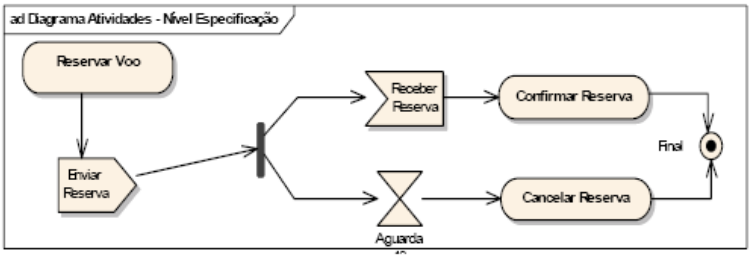
- Diagrama Atividades em nível de Especificação
  - Os diagramas abaixo mostram como podem ser utilizados sinais em um diagrama de atividade.
  - Um sinal pode ser enviado, recebido, ou ainda ser um sinal que indica algum momento no tempo
  - As junções (joint) e bifurcações (fork) podem ter associadas às mesmas critérios (joinspec) indicando quando a junção ou bifurcação pode acontecer

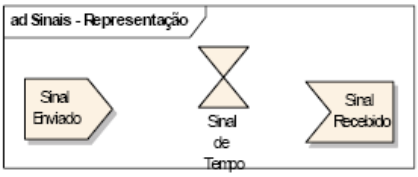




UML – Diagramas Comportamentais
Diagrama de Atividades - Ações

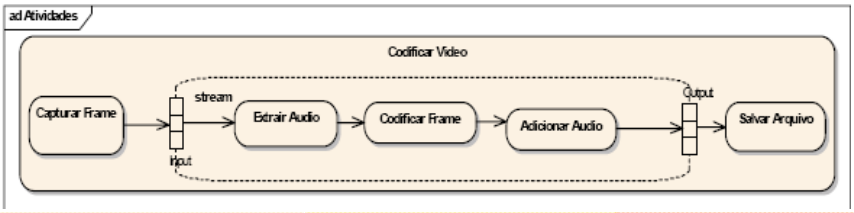
- Modelamento de Sinais (Estado de Ação)





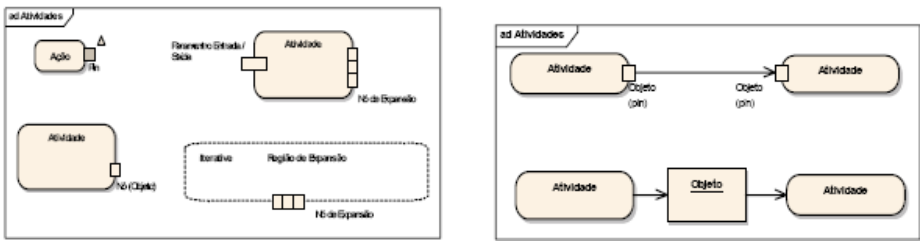
UML – Diagramas Comportamentais
Diagrama de Atividades – Reg. Expansão

- Regiões de Expansão
  - Indicam atividades que serão executadas para cada item de uma coleção de valores ou objetos
  - A execução pode ser:
  - iterativa(**iteractive) – Execução das atividades são feitas de forma seqüencial**
  - Paralela (**parallel) – Execução das atividades podem ocorrer de forma concorrente**
  - Fluxo (**Stream) – Execução das atividades acontece de forma contínua**



UML – Diagramas Comportamentais
Diagrama de Atividades - Sinais

- É possível indicar a existência de **parâmetros de entrada e de saída em uma** atividade
- Além disso é possível indicar **nós de expansão que representam uma coleção** como parâmetro de entrada e/ou saída de uma região de expansão ou atividade. Esta coleção será então utilizada por outra atividade
- Finalmente uma atividade pode possuir um **nó objeto, também chamado pin.** O **Pin representa um objeto que é enviado de uma atividade para outra e** consiste de uma outra forma de representar o fluxo de objeto



Praticando seus conceitos...

- Considere um sistema na Web que será responsável por gerenciar contatos, conforme caso de uso anterior
  - Além de cadastrar os contatos é possível; consultar os contatos; alterar suas informações; imprimir seus dados e finalmente enviar e-mail para um determinado contato
- Informações adicionais
  - Um usuário pode possuir vários contatos e o sistema deverá manter os dados de cada usuário individualmente
  - Um contato pode possui até 4 diferentes endereços de e-mail e para cada e-mail está associado um tipo (comercial; particular; prioritário; final de semana; etc.)
  - As informações associadas ao contato são as seguintes: Nome; Telefone Principal; Email
- Construir os diagramas de classe em nível de análise
- Construir um diagrama de classe que representa a modelagem de dados
- Construir os diagramas de atividade. Cada atividade será então realizada por um diagrama de seqüência, neste caso o interesse são nos objetos envolvidos

Praticando seus conceitos... Sistema de Pedidos

- Informações Gerais
  - Existem 3 níveis de usuário: Supervisor; Gerente e Diretor, com acesso a diferentes funcionalidades
  - Supervisor pode cadastrar pedidos. O gerente aprova pedidos deste que o mesmo não ultrapasse o valor total associado ao cliente para cada pedido. Caso ultrapasse o diretor é o responsável pela aprovação.
  - Um pedido é associado a um cliente e o mesmo contém os itens de pedido. Cada item de pedido contém: Produto; Quantidade; Valor total.
  - Caso não seja aprovado o pedido poderá também ser cancelado.
  - Após realizar o login na página principal são mostradas os pedidos que necessitam de aprovação.
  - Um supervisor tem sob sua responsabilidade um ou mais clientes.
- Pede-se:
  - Criar os Diagramas de Casos de Uso
  - Modelar as classes considerando os conceitos de: Interface; Polimorfismo; Sobrecarga
  - Construir os diagramas de atividade que demonstre os fluxos principais do sistema.
  - Criar o diagrama de seqüência para os fluxos de aprovação, levando em conta os objetos de negócio envolvidos.