Projeto Detalhado

Projeto Orientado a Objetos

Projeto Detalhado

- Introdução ao projeto detalhado OO
- Construindo os digramas de interação
- Padrões GRASP
- Diagrama de Classes – Perspectivas de Projeto

Passos de Projeto

- **Projeto Geral ou Preliminar**: fase que traduz a especificação do sistema em termos da arquitetura de dados e de módulos
- **Projeto Detalhado**: refinamento visando à codificação e especificação dos programas

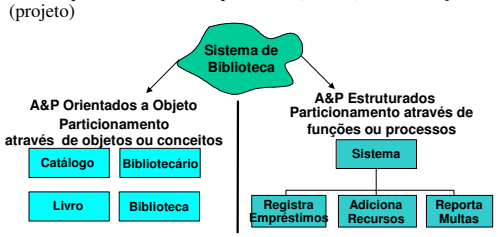
Modelo de decomposição em módulos

Especifica a decomposição de cada sub-sistema em módulos

- **Orientados a funções**: os sub-sistemas são decompostos em módulos funcionais que recebem dados e transformam estes dados em saída
- **Orientado a objetos**: os sub-sistemas são decompostos em um conjunto de objetos que se comunicam para resolver o problema (ex:diagramas de interação -UML)

Análise e Projeto OO

- O que determina uma orientação a objetos é a maneira como se faz o particionamento do problema (análise) ou da solução (projeto)

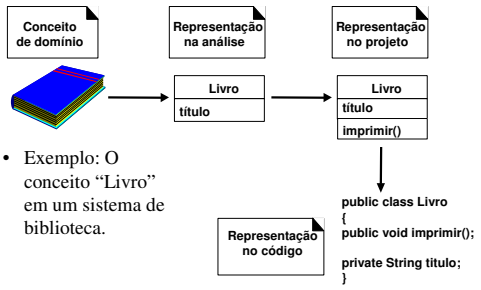


Análise e Projeto OO

- Durante a Análise OO, a ênfase está em achar e descrever objetos (ou conceitos) no domínio do problema

- Durante o projeto OO, a ênfase está em achar objetos lógicos de software que poder ão ser eventualmente implementados usando uma linguagem OO

Análise e Projeto OO



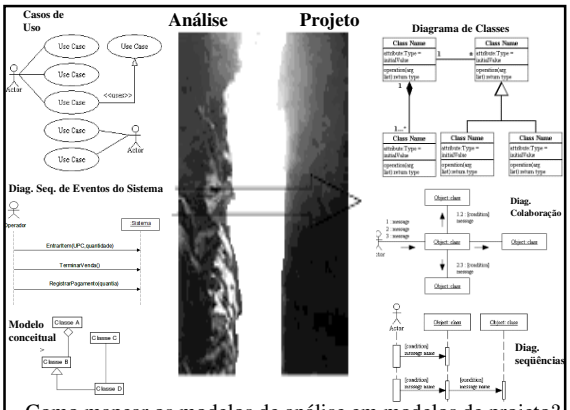
- Exemplo: O conceito "Livro" em um sistema de biblioteca.

Ferramentas para criar sw OO

- Saber uma linguagem de programação orientada a objeto (OO) não é suficiente para criar sistemas OO
- É preciso saber Análise e Projeto OO
- Linguagem UML
- Padrões de projeto

Atividades do projeto OO

- Atribuição de responsabilidades entre os objetos
- Construção de diagramas de classes
- Construção de diagramas de interação (seqüência e colaboração)
- Levantamento de necessidades de concorrência
- Considerações de tratamento de defeitos
- Detalhamento do formato de saída (interface com usuário, relatórios, transações enviadas para outros sistemas, ...)
- Definição do esquema do BD: mapeamento de objetos para tabelas se o BD for relacional



Como mapear os modelos de análise em modelos de projeto?

Mapeando os modelos de análise para os de projeto

- Artefatos de análise capturam os resultados do processo de investigação do domínio do problema

Casos de Uso	Quais são os processos do domínio?
Modelo Conceitual	Quais são os conceitos, termos, etc.?
Diagramas de Seqüência de Eventos do Sistema	Quais são os eventos e operações do sistema?
Contratos	O que fazem as operações do sistema?
Cartões CRC	Quais são as classes, responsabilidades e, colaborações

Mapeando os modelos de análise para os de projeto

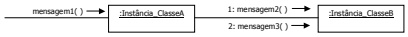
- A partir dos artefatos da fase de análise é desenvolvida uma solução lógica.
- Os Diagramas de Interação (Seqüência e/ou Colaboração) são a base de tal solução, a partir deles, posteriormente, são criados os Diagramas de Classes (de projeto)
- ***"Identificado os requisitos e o modelo conceitual, acrescente os métodos às classes de software e defina as mensagens/troca de mensagens (nos DI) para atender aos requisitos" Larman??***

Construindo os diagramas de interação

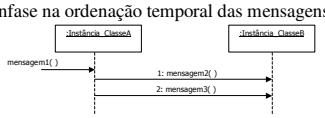
Diagramas de Interação

Ilustram como os objetos interagem através de mensagens para cumprir suas tarefas.

- Diagramas de Colaboração
  - Ênfase na organização estrutural dos objetos que enviam e recebem mensagens



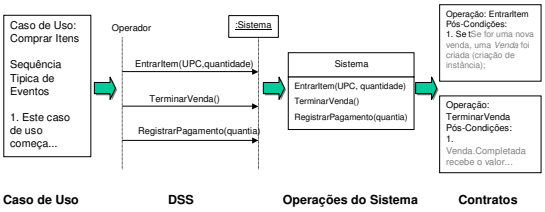
- Diagramas de Seqüência
  - Ênfase na ordenação temporal das mensagens:



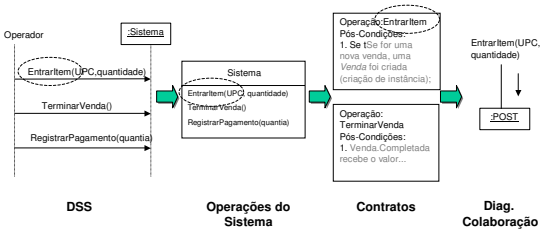
Artefatos de análise como ponto de partida

- Modelo Conceitual: subsidia a definição de classes de software correspondentes a conceitos, cujos objetos participam dos DI
- Contratos: subsidia a definição de responsabilidades e as pós-condições que os DI devem satisfazer
- Casos de Uso : subsidia a coleta de informações sobre quais tarefas os DI ilustram, além do que está nos contratos
- Diagrama de seqüências de eventos do sistema (DSS): mostra para um dado caso de uso, os eventos aos quais o sistema deve responder. Para responder a um evento o sistema deve implementar uma operação correspondente, que será executada como resposta ao evento recebido.

Relação entre os artefatos de análise



Artefatos de Análise e Diagramas de Interação



Como Fazer um Diagrama de Interação

- Regras úteis:
  1. Criar um diagrama em separado para cada uma das operações de sistema em desenvolvimento.
     - Para cada mensagem de operação de sistema, criar um diagrama com essa mensagem como mensagem inicial.
  2. Se um diagrama se tornar muito complexo, o diagrama pode ser dividido.
  3. Utilizar as responsabilidades e pós-condições dos contratos das operações de sistema, ou dos CRCs, e a descrição dos casos de uso para projetar um sistema cujo objetos interajam para cumprir tarefas.
  4. Utilizar padrões de projeto

Responsabilidades

- Responsabilidade é "um contrato ou obrigação de um tipo ou classe" (Booch e Rumbaugh, 1997)
  - Relacionada às obrigações dos objetos em termos de comportamento.
- Tipos básicos
  - **saber**: saber sobre os seus dados, sobre objetos relacionados, sobre coisas que ela pode calcular ou derivar.
  - **fazer**: iniciar ações entre outros objetos, controlar e coordenar atividades em outros objetos

Responsabilidades e Métodos

- Responsabilidades são atribuídas aos objetos do sistema durante o Projeto OO
  - "Uma *Venda* é responsável por imprimir a si própria" (de fazer)
  - "Uma *Venda* é responsável por conhecer sua data" (de conhecer)
- Tradução de responsabilidades para classes e métodos é influenciada pela granularidade da responsabilidade
  - Um único método para "imprimir venda"
  - Dezenas de métodos para "prover um mecanismo de acesso a SGBDR"

Responsabilidades e Métodos

- Métodos são implementados para cumprir responsabilidades
  - Uma responsabilidade não é igual a um método.
  - Podem agir sozinhos ou em colaboração com outros métodos e objetos
- Exemplo:
  - A classe *Venda* pode definir um ou mais métodos específicos para cumprir a responsabilidade de *imprimir*
  - Esse método, por sua vez, pode precisar colaborar com outros objetos, possivelmente enviando mensagens de impressão para cada um dos objetos *ItemVenda* associados à *Venda*.

Padrões GRASP

(**G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns)

Conceito de Padrão

- Um template (formulário) de solução para um problema recorrente que seja comprovadamente útil em um determinado contexto.
- Um padrão de software é instanciado através da vinculação de valores a seus parâmetros.
- Os padrões podem existir em várias escalas e níveis de abstração; por exemplo, como *padrões de arquitetura*, *padrões de análise*, *padrões de projeto*, *padrões de teste* e *idiomas* ou *padrões de implementação*.
- Padrão genérico

Histórico

- Arquiteto ->Christopher Alexander –
- Linguagem de padrões em arquitetura.
  - catálogo com 253 padrões para edificações ligadas a regiões, cidades, transportes, casas, escritórios, paredes, jardins, etc.

Definição

- "um padrão expressa uma solução reutilizável descrita através de três partes: um contexto, um problema e uma solução". (GAMMA et al., 1995).
- **Contexto**: estende o problema a ser solucionado, apresentando situações de ocorrência desses problemas.
- **Problema:** determinado por um sistema de forças, onde estas forças estabelecem os aspectos do problema que devem ser considerados.
- **Solução**: mostra como resolver o problema recorrente e como balancear as forças associadas a ele.

Padrões em ES

- Padrões em ES permitem que desenvolvedores possam recorrer a soluções já existentes para solucionar problemas que normalmente ocorrem em desenvolvimento de software;
- Surgimento: início dos anos 90;
  - 1995 - livro da "Gang of Four" (GoF)
    - 23 padrões de projeto (design patterns)
  - OOPSLA
- Padrões capturam experiência existente e comprovada em desenvolvimento de software, ajudando a promover boa prática de projeto.

Padrões GRASP

(**G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns)

- Codificam idéias e heurísticas existentes para **atribuir responsabilidades** a objetos.
- Auxiliam a elaborar os diagramas de interação.

Padrões GRASP básicos: especialista, criador, alta coesão, baixo acoplamento, controle.

1. Padrão Especialista (Expert)

Problema: : Qual é o princípio básico pelo qual responsabilidades são atribuídas em POO?

Solução: : Atribua uma responsabilidade a uma classe que possui a informação necessária para cumprí-la.

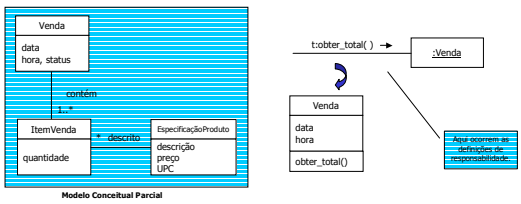
Exemplo: Quem deveria ser responsável por calcular o total-geral de uma venda?

a classe Venda possui a informação para isso

1. Padrão Especialista (Expert)

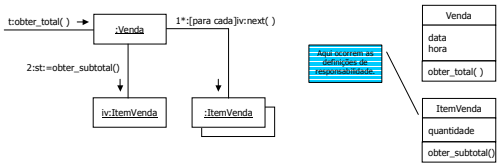
Classe	Responsabilidade
Venda	sabe total geral da venda
Item de Venda	sabe sub-total de cada item
Especificação do Produto	sabe preço do produto

1. Padrão Especialista (Expert)



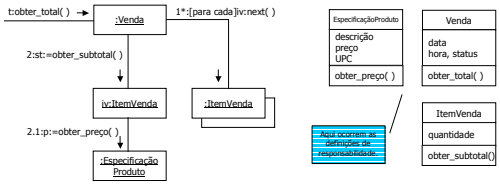
1. Padrão Especialista (Expert)

- Mas quem deve ser responsável por conhecer o **subtotal** de um *ItemVenda*?
  - Informação necessária: *ItemVenda.quantidade* e *EspecificaçãoProduto.preço*
  - Pelo padrão, a classe *ItemVenda* deve ser a responsável.



1. Padrão Especialista (Expert)

- Porém, para cumprir essa responsabilidade de conhecer ou informar seu subtotal um *ItemVenda* precisa conhecer o preço do Item.
  - Portanto, o *ItemVenda* deve mandar uma mensagem para a EspecificaçãoProduto para saber o preço do item.



1. Padrão Especialista (Expert)

- É o padrão mais usado de todos para atribuir responsabilidades
- Conhecido como:
  - "Colocar as responsabilidades com os dados"
  - "Quem sabe, faz"
  - "Animação"
  - "Eu mesmo faço"
  - "Colocar os serviços junto aos atributos que eles
  - manipulam"

1. Padrão Especialista (Expert)

- A informação necessária freqüentemente está espalhada em vários objetos, mensagens são usadas para estabelecer colaborações
- Portanto, tem muitos experts parciais
  - Exemplo: determinar o total de uma venda requera colaboração de 3 objetos, em 3 classes diferentes
- O resultado final é diferente do mundo real
  - Atribuir responsabilidades é dar vida aos objetos/classes.
  - No mundo real, uma venda não calcula seu próprio total. Isso seria feito por uma pessoa (se não houvesse software).

1. Padrão Especialista (Expert)

- Encapsulamento é mantido, já que objetos usam sua própria informação para cumprir suas responsabilidades
- Leva a fraco acoplamento entre objetos e sistemas mais robustos e fáceis de manter
- Leva a alta coesão, já que os objetos fazem tudo que é relacionado à sua própria informação

Exercício: melhore o diagrama abaixo usando o padrão Expert.

2. Padrão Criador (Creator)

Problema: Quem deve ser responsável por criar uma nova instância de alguma classe?

Solução: Atribua à classe B a responsabilidade de criar uma instância da classe A se:

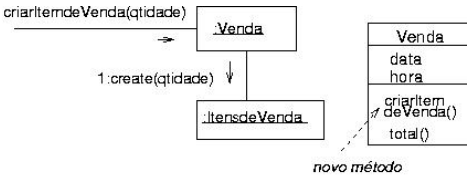
1. B agrega objetos de A
2. B contém A
3. B armazena instâncias de A
4. B usa objetos de A
5. B possui informação necessária a criação de A (B é um especialista para criar A)

Se mais de uma opção se aplica, escolha o B que agregue ou contenha objetos da classe A

2. Padrão Criador (Creator)

Exemplo: Quem deveria ser responsável por criar a instância da classe Item de Venda? *classe Venda agrega muitos objetos da classe Itens de Venda*

2. Padrão Criador (Creator)



Para Criar uma Linha de Item de Venda

2. Padrão Criador (Creator)

- Escolher um criador que estará conectado ao objeto criado, de qualquer forma, depois da criação leva a fraco acoplamento, o objeto precisa ser visível ao criador depois da criação de qualquer maneira mesmo.
- Exemplo de criador que possui os valores de inicialização
  - Uma instância de Pagamento deve ser criada
  - A instância deve receber o total da venda
  - Quem tem essa informação? Venda
  - Venda é um bom candidato para criar objetos da classe Pagamento

3. Padrão Baixo Acoplamento e (Low Coupling)

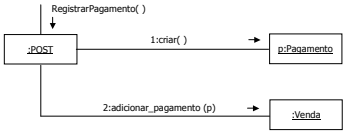
Problema: Como minimizar dependências e maximizar o reuso?

Solução: Atribuir a responsabilidade de modo que o acoplamento (dependência entre classes) seja baixo.

O acoplamento é uma medida de quão fortemente uma classe está conectada, possui conhecimento ou depende de outra classe
Com forte acoplamento, temos os seguintes problemas:
- Mudanças em uma classe implica mudanças na classe dependente
- A classe é mais difícil de entender isoladamente
- A classe é mais difícil de ser reusada

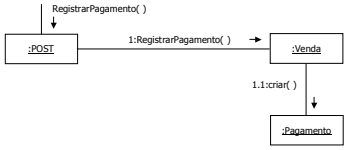
3. Padrão Baixo Acoplamento e (Low Coupling)

- Exemplo
  - Quem deve ser responsável por criar um Pagamento e associá-lo à Venda?
  - Pelo padrão Criador, poderia ser POST (uma vez que POST "registra" pagamentos no mundo real)



3. Padrão Baixo Acoplamento e (Low Coupling)

- Uma solução melhor, do ponto de vista do padrão, que preserva baixo acoplamento é a própria Venda criar Pagamento, pois Venda tem que ter conhecimento de Pagamento



3. Padrão Baixo Acoplamento e (Low Coupling)

- Benefícios
  - Responsabilidade não é (ou é pouco) afetada por mudanças em outros componentes.
  - Responsabilidade é mais simples de entender isoladamente.
  - Maior chance para reuso.

3. Padrão Baixo Acoplamento e (Low Coupling)

- Acoplamento se manifesta de várias formas:
  - X tem um atributo que referencia uma instância de Y
  - X tem um método que referencia uma instância de Y
  - Pode ser parâmetro, variável local, objeto retornado pelo método
  - X é uma subclasse direta ou indireta de Y
  - X implementa a interface Y
  - A herança é um tipo de acoplamento particularmente forte
- Não se deve minimizar acoplamento criando alguns poucos objetos "deuses" (God classes)
- Exemplo: todo o comportamento numa classe e outras classes usadas como depósitos passivos de informação

3. Padrão Baixo Acoplamento e (Low Coupling)

- Tipos de acoplamentos (do menos ruim até o pior)
  - Acoplamento de dados
  - Acoplamento de controle
  - Acoplamento de dados globais
  - Acoplamento de dados internos (pior de todos)

4. Padrão Alta Coesão (High Cohesion)

Problema: Como manter a complexidade (das classes) sob controle?

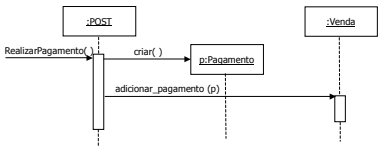
Solução Atribuir uma responsabilidade de modo que a coesão seja alta

A coesão é uma medida do quanto as responsbilidades de uma classe estão relacionadas.

- Alta: A Classe tem uma responsabilidade bem definida e faz poucas coisas
- Baixa: Uma classe faz muitas coisas não relacionadas ou executa muitas tarefas (indesejável pois fica difícil de: compreender, reutilizar e manter; e são constantemente afetadas pelas mudanças)

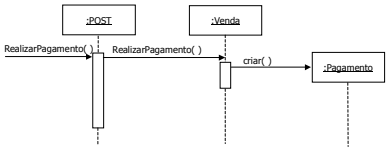
4. Padrão Alta Coesão (High Cohesion)

- Exemplo
  - Quem deve ser responsável por criar um Pagamento e associá-lo à Venda?
  - Pelo padrão Criador, seria POST. Mas se POST for o responsável pela maioria das operações do sistema, ele vai ficar cada vez mais sobrecarregado e sem coesão.



4. Padrão Alta Coesão (High Cohesion)

- Exemplo
  - Outra forma para atribuição da responsabilidade RealizarPagamento que favorece uma coesão mais alta



4. Padrão Alta Coesão (High Cohesion)

- Níveis de coesão
  - Muito baixa
    - Um única classe é responsável por muitas coisas em áreas funcionais muito diferentes.
  - Baixa
    - Um classe é a única responsável por uma tarefa complexa em uma área funcional.
  - Alta
    - Um classe tem responsabilidades moderadas em uma área funcional e colabora com outras classes para realizar tarefas.
  - Moderada
    - Uma classe tem peso leve e responsabilidades exclusivas em algumas áreas logicamente relacionadas ao conceito da classe, mas não uma com a outra.

4. Padrão Alta Coesão (High Cohesion)

- Benefícios
  - Aumento da clareza e compreensão do projeto
  - Simplificação da manutenção
  - Favorece baixo acoplamento
  - Reuso facilitado

5. Padrão Controlador (Controler)

Problema: Quem deveria ser responsável por lidar com um evento do sistema?

um controlador é um objeto de interface responsável por gerenciar um evento do sistema, definindo métodos para operações do sistema

Solução: Atribua responsabilidades para lidar com mensagens de eventos do sistema a uma classe que:

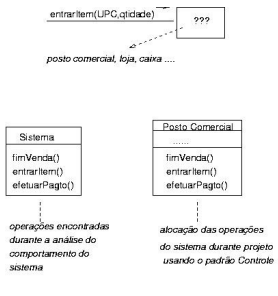
1. represente o sistema como um todo
2. represente a organização
3. represente algo ativo no mundo real envolvido na tarefa (por exemplo o papel de uma pessoa)
4. represente um controlador artificial dos eventos de sistema de um caso de uso

Diagrama de Colaboração – Padrões Grasp

Exemplo: Quem deveria ser o controlador para eventos do sistema tais como entrarItem e Vendafim?

5. Padrão Controlador (Controler)

Exemplo: Quem deveria ser o controlador para eventos do sistema tais como entrarItem e Vendafim?



Exemplificando o Uso do Padrão Controle

5. Padrão Controlador (Controler)

- Use o mesmo controlador para todos os eventos do sistema no mesmo caso de uso
- Classes do tipo janela, applet, aplicações, documento não deveriam realizar tarefas associadas a eventos do sistema. Elas apenas recebem e delegam os eventos ao controlador
  - Um evento do sistema é gerado por um ator

5. Padrão Controlador (Controler)

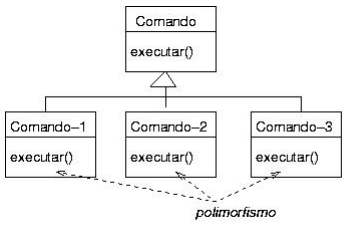
- Um controlador não deve ter muitos atributos e nem manter informação sobre o domínio do problema
- Um controlador não deve realizar muitas tarefas, apenas delegá-las
- Um bom projeto deve dar vida aos objetos, atribuindo-lhes responsabilidades, até mesmo se eles forem seres inanimados
- A camada de apresentação (interface com o usuário) não deve tratar eventos do sistema

6. Padrão Comando (Command)

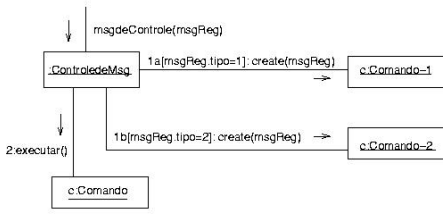
Problema: Aplicações que recebem msg de um sistema externo (não existe interface com o usuário). Ex: sistemas de telecomunicações

Solução: definir uma classe para cada comando, com o método executar. O controlador criará uma instância da classe correspondente a msg do evento do sistema e enviará a mensagem executar à classe comando.

6. Padrão Comando (Command)



Usando Padrão Comando e Controlador



Usando os Padrões Comando e Controlador

Diagramas de Interação: Como Construir

- utilize as responsabilidades e pós-condições dos contratos e use casos de usos como ponto de partida
- escolha a classe que controlará o sistema, aplicar padrão controle
- para cada operação existe um contrato, uma operação vai ser uma mensagem.

Diagramas de Interação: Como Construir

- separação do modelo de visão: não é responsabilidade dos objetos do domínio se comunicarem com o usuário (ignorar responsabilidades de apresentação dos dados no display, mas toda informação do domínio de objetos tem que ser mostrada)
- para um objeto enviar uma msg a outro é necessário **visibilidade**: habilidade de um objeto ver ou fazer referência a outro objeto

Diagrama de Colaboração – Exemplos - Padrões GRASP

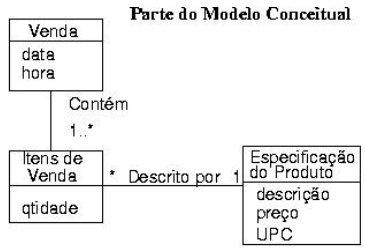


Diagrama de Colaboração – Exemplos - Padrões GRASP

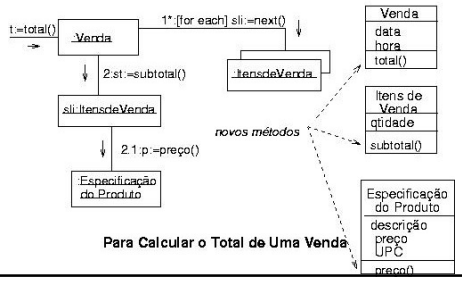


Diagrama de Colaboração – Exemplos - Padrões GRASP

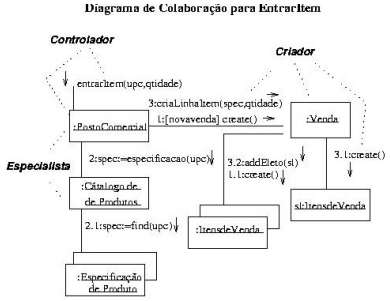


Diagrama de Colaboração – Exemplos - Padrões GRASP

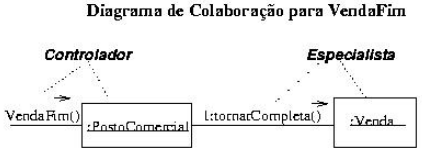


Diagrama de Colaboração – Exemplos - Padrões GRASP

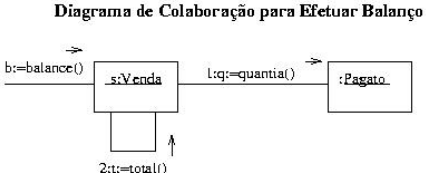


Diagrama de Colaboração – Exemplos - Padrões GRASP

Diagrama de Colaboração para Registrar Venda

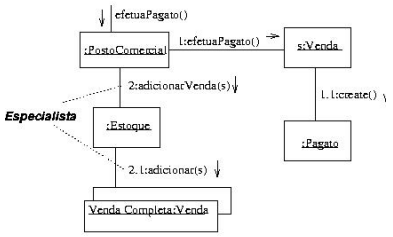


Diagrama de Colaboração – Exemplos - Padrões GRASP

Diagrama de Colaboração para Calcular o Total de Venda

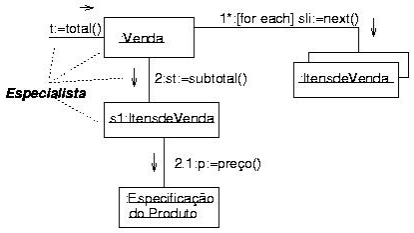


Diagrama de Colaboração – Exemplos - Padrões GRASP

Diagrama de Colaboração para Efetuar Pagamento

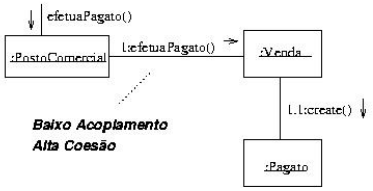


Diagrama de Classes

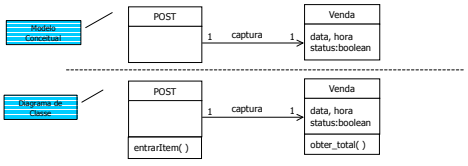
Modelo de classes de especificação
Perspectiva de Projeto

Diagrama de Classes

- Ilustra as especificações de software para as classes e interfaces do sistema.
- É obtido através da adição de detalhes ao modelo conceitual conforme a solução de software escolhida, acrescentando-se:
  - classes de controle e de interface (fronteira);
  - métodos (tipos de parâmetros e de retorno).
  - visibilidade e navegabilidade,
  - inclui dependências,
  - inclui a visão de projeto além da visão de domínio

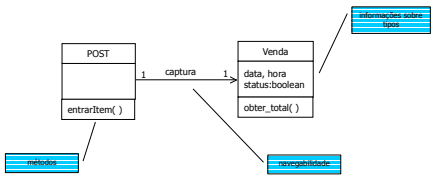
Modelo de Conceitual X Diagrama de Classe

- **Modelo Conceitual**: abstração de conceitos do mundo real
- **Diagrama de Classe**: especificação de componentes de software



Diagramas de Classe

- Diagrama parcial para as classes **POST** e **Venda** no sistema POST:



Como construir (1)

- identificar todas as classes participantes da solução através dos diagramas de interação
- desenhar o diagrama
- copiar os atributos
- adicionar os métodos dos diagramas de interação
- adicionar tipos de atributos, parâmetro e retornos de métodos.

Como construir (2)

- adicionar as associações necessárias a visibilidade
- adicionar navegabilidade que indica a direção de visibilidade por atributo
- adicionar setas pontilhadas indicando visibilidade que não é por atributo
- Métodos ``create" e de acessos aos atributos podem ser omitidos.
- Os tipos podem ou não ser mostrados; classes podem ser detalhadas ao máximo

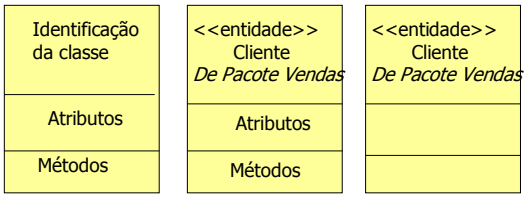
Objetos

- Perspectiva de implementação: representa um módulo de sw que recebe e produz dados
  - Identidade – identificador em lg de implementação
  - Atributos – variáveis e seus tipos, que recebem diferentes valores e definem o estado do objeto
  - Comportamento – funções ou procedimentos, os resultados dessas funções determinam o comportamento do objeto

Classes

- Perspectiva de implementação: corresponde a um tipo de uma lg de programação
- Um modelo genérico para criar variáveis que armazenarão os objetos correspondentes.

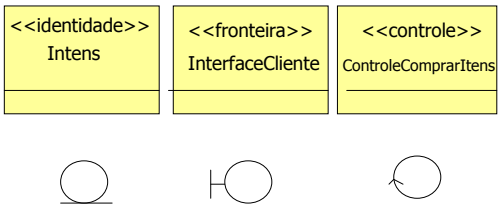
Notação UML para Classes



Identificação de Classes com Estereótipos

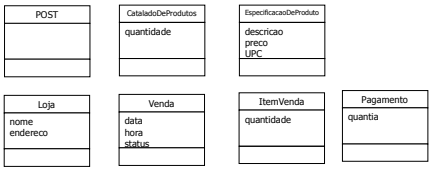
- Estereótipo é um classificador. Tipos:
- **Entidade**: representam conceitos do mundo real e armazenam dados que os identificam – como no modelo conceitual
- **Controle**: controlam a execução de processos e o fluxo de execução de todo ou de parte de casos de uso e comandam outras classes
- **Fronteira:** realizam o interfaceamento com entidades externas (atores), contém o protocolo de comunicação com impressora, monitor, teclado, disco, porta serial, modem, etc.

Exemplos no Sistema Posto Comercial



Identificação de Classes Entidades – Exemplos sistema POST

- Identificando classes e atributos
  - Observar os DI e Modelo Conceitual



Identificação das Classes de Controle

- Definir pelo menos uma classe do tipo controle para cada caso de uso de forma que ela contenha a descrição e comando do processo associado ao caso de uso.
- Definir classes de controle auxiliares em certos casos de uso que devido aà complexidade requeiram a divisão de seu processo em subprocessos. As classes auxiliares seriam controladas pela classe de controle principal

Identificação das Classes de Controle

- Suas principais características são:
  - Cria, ativa e anula objetos controlados
  - Controla a operação de objetos controlados
  - Controla a concorrência de pedidos de objetos controlados
  - Em muitos casos corresponde a implementação de um objeto intangível.
- Gerente de Registro para o Caso de Uso registrarAlunos

Identificação das Classes de Fronteira

- Definir uma classe do tipo fronteira para cada ator que participe do caso de uso, pois cada ator que pode exigir um protocolo próprio para comunicação.
- Uma classe para interface com o usuário, uma classe para interface com a impressora, uma classe para interface com a porta serial, etc.

Identificação das Classes de Fronteira

- Exemplos: Interface tipo Janela, Protocolo de Comunicação, Interface de Impressão, Sensores, etc.
  - Classes: Formulário em Branco e Sistema de Cobrança

Atributos – refinando os tipos Notação UML

- A maioria é opcional, seu uso vai depender do tipo de visão no qual estamos trabalhando e podem ser abstratos ou utilizar a notação de uma lg de programação

[Visibili/d]Nome[Multiplici/d]:[Tipo]=[Valor][{Proprie/ds}]

Notação UML para Atributos - Visibilidade

- **+** : visibilidade pública: o atributo ü visível no exterior da classe.
- **-** : visibilidade privada : o atributo é visível somente por membros da classe.
- **#** : visibilidade protegida: o atributo é visível também por membros de classes derivadas

Notação UML para Atributos - Multiplicidade

- Usada para especificar atributos que são arranjos
- Indica dimensão de vetores e matrizes
  - Ex: notas[10]
  - matrizDeValores[5,10]

Notação UML para Atributos - Tipos

- Indicam o formato do valores que o atributo pode assumir
- Na visão conceitual o tipo é abstrato

  Ex: dataDaVenda: tipoData
- Na visão de implementação utilizam-se os tipos da lg de programação

  Ex: salario: float

Notação UML para Atributos – Valor Inicial e Propriedades

- Pode-se indicar o valor ou conteúdo do atributo imediatamente após a sua criação, ou o seu valor default

  Ex: resultado: int=0
- As propriedades descrevem comentários ou indicações sobre o atributo, podem mostrar se ele é ou não opcional

  Ex: dataDaVenda {valor constante}

Identificação dos Métodos

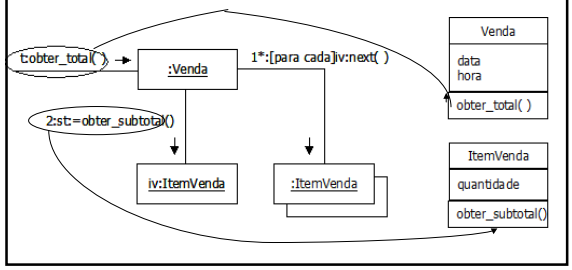
- Os métodos são acrescentados nas perspectivas de especificação e de implementação e são derivados a partir dos diagramas de interação: colaboração e sequências, na fase de projeto detalhado.
- É útil distingüir operações de métodos. Operações é algo que se evoca sobre um objeto (a chamada do procedimento). Para realizar uma operação a classe implementa um método (o corpo do procedimento)

Identificação dos Métodos

- É útil distingüir operações que alteram ou não o estado (atributos) de uma classe
- Não alteram: query, métodos de obtenção, getting methods
- Alteram: modificadores, métodos de atribuição ou fixação, setting methods

Métodos a partir dos DI

- Se uma classe recebe uma mensagem A, ela deverá executar em resposta uma operação A que é implementada por um método A.



Métodos

- Os métodos são acrescentados na perspectiva de implementação e são derivados a partir dos diagramas de interação: colaboração e sequências.
- É útil distingüir operações de métodos. Operações é algo que se evoca sobre um objeto (a chamada do procedimento). Para realizar uma operação a classe implementa um método (o corpo do procedimento)

Métodos

- É útil distingüir operações que alteram ou não o estado (atributos) de uma classe
- Não alteram: query, métodos de obtenção, getting methods
- Alteram: modificadores, métodos de atribuição ou fixação, setting methods

Notação UML para Métodos

- A notação e uso vai depender do tipo de visão no qual estamos trabalhando e podem ser abstratos ou utilizar a notação de uma lg de programação

[Visibili/d]Nome(Parâmetros)]:[Retorno][{Proprie/ds}]

Notação UML para Métodos - Parâmetros

- Os parâmetros – dados de entrada e/ou saída para o método são representados por

  Nome-do-Parâmetro:Tipo=Valor-Padrão

Ex:

- Visão conceitual

  ImprimirData(data:TipoData)

Visão de implementação

- ArmazenarDados(nome:char[30],salario:float=0.0)

Notação UML para Métodos – Tipo de Retorno

- O Valor-de-Retorno indica se o método retorna algum valor ao término de sua execução e qual o tipo de dado do valor retornado.

Ex:

- Visão Conceitual

  CalcularValor(): TipoDinheiro

- Visão de implementação

  ArmazenarDados(nome:char[30]): bool

Notação UML para Métodos – Visibilidade e Propriedades

- Visibilidade – similar ao de atributo

- Comentários ou restrições para os métodos
- Ex:

  Area() {Área <=600}

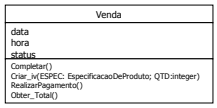
  Restringe a 600 unidades o valor máximo das áreas a calcular

Notação UML para Métodos – Propriedades

- É útil distingüir operações de métodos. Operações é algo que se evoca sobre um objeto (a chamada do procedimento). Para realizar uma operação a classe implementa um método (o corpo do procedimento)
- É útil distingüir operações que alteram ou não o estado (atributos) de uma classe
- Não alteram: query, métodos de obtenção, getting methods
- Alteram: modificadores, métodos de atribuição ou fixação, setting methods

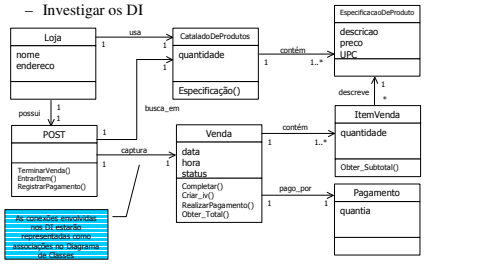
Criando Diagramas de Classe para o Sistema POST

- Adicionando informação sobre o tipo dos atributos
  - Opcional
  - Grau de detalhe dependente do público-alvo.



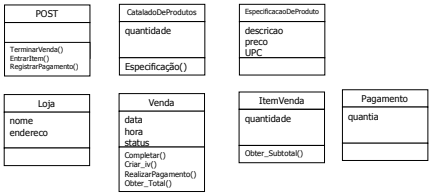
Criando Diagramas de Classe para o Sistema POST

- Adicionando associações e navegabilidade
  - Investigar os DI



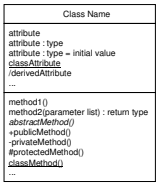
Criando Diagramas de Classe para o Sistema POST

- Adicionando nomes aos métodos
  - Observe as mensagens dos DI



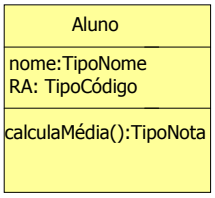
Características dos Elementos de Classe

- UML oferece notação rica para descrever características como visibilidade, valores iniciais, etc.
- No sistema POST: todos os atributos são privados e todos os métodos são públicos

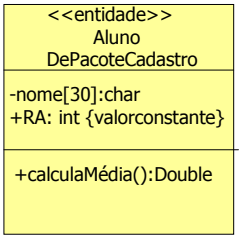


Exemplo de Uma Classe

Visão Conceitual



Visão de Implementação



Outros aspectos

Visibilidade

Habilidade de um objeto A ver ou fazer referência a um outro objeto B. Para A enviar uma msg para B, B deve ser visível a A.

- por atributo: B é um atributo de A
- por parâmetro: B é um parâmetro de um método de A
- localmente declarada: B é um objeto local de um método de A; uma instância local é criada, ou ele será um valor de retorno
- global: B é visível globalmente; usa-se uma variável global para armazenar uma instância - pouco recomendada

Diagrama de Colaboração - Visibilidade

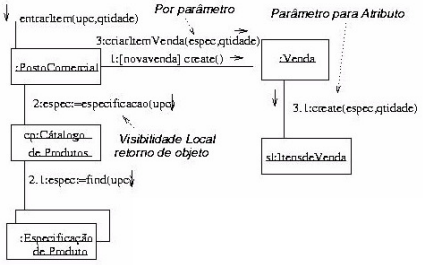


Diagrama de Colaboração - Visibilidade

```
PostoComercial .... entrarItem(upc,qtididade)
{
   ...
   espec=cp.especificacao(upc);
   ...
}
```

```
Class PostoComercial
{
   ...
   private CatalogodeProdutos cp;
   ...
}
```

```
Venda ... criarItemVenda(espec,qtidade)
{
   ...
   si=new ItensdeVenda(espec,qtidade)
   ...
}
```

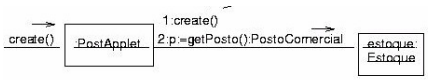
```
construtor ...ItensdeVenda(espec,qtidade)
{
   ...
   prod=espec;
   ...
}
```

Diagrama de Colaboração - Inicializando

Como se realizam as operações iniciais da aplicação?

- Cria-se um caso de uso startUp.
- Seu diagrama de colaboração deve ser criado por último, representando o que acontece quando o objeto inicial do problema é criado.
- Quem deveria ser o objeto inicial do sistema?
  - classe representando toda a informação lógica do sistema
  - classe representando a organização
  - usar Padrões Alta Coesão e Baixo Acoplamento

Diagrama de Colaboração - Inicializando




```
public class PostApplet extends Applet
public void init()
{
   p=estoque.getPosto
}
```

estoque é o objeto inicial do domínio
e o construtor Store cria os outros objetos.

Diagrama de Colaboração - Inicializando

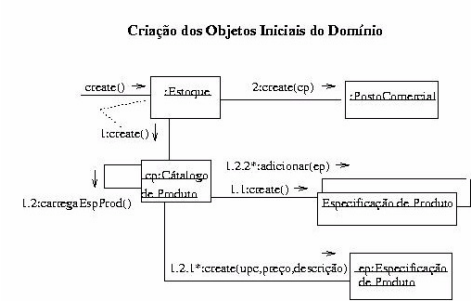


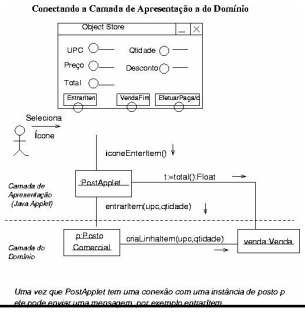
Diagrama de Colaboração - Inicializando

- Conectando a camada de apresentação com a do domínio
- Uma operação de startUp pode ser:
  - uma mensagem ``create" para o objeto inicial;
  - se o objeto inicial é o controlador, uma mensagem ``run" para um objeto inicial é enviada.

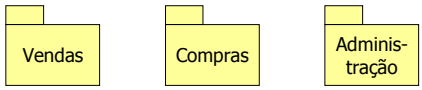
Diagrama de Colaboração - Inicializando

- Se uma interface do usuário estiver envolvida ela é responsável por iniciar a criação do objeto inicial e outros associados.
- Objetos da camada de apresentação não devem ter responsabilidades lógicas. Das nossas escolhas resultarão extensibilidade, claridade e manutenibilidade

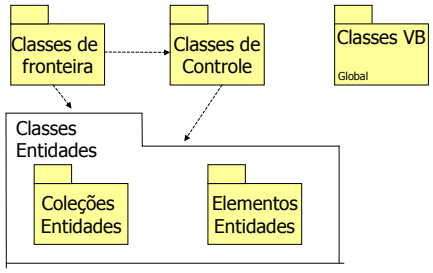
Diagrama de Colaboração - Inicializando



Organização de Classes em Pacotes Lógicos



Organização de Classes em Pacotes Lógicos



Referências

- Boock, G. and Rumbaugh, J. *The Unified Modeling Language User Guide* . Addison-Wesley, 1999
- Arlow, J. and Neustadt, I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, 2nd Edition, The Addison-Wesley Object Technology Series, 2005.
- Rumbaugh, J.; Jacobson, I. and Booch , G. The Unified Modeling Language Reference Manual, 2nd Edition, The Addison-Wesley Object Technology Series, 2004.
- Boock, G.; Rumbaugh, J. and Jacobson, I; Unified Modeling Language User Guide, 2nd Edition, The Addison-Wesley Object Technology Series, 2005.
- Jacobson, I; Boock, G. and Rumbaugh, J., Unified Software Development Process, Addison-Wesley, Janeiro 1999.
- Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* Prentice-Hall, New Jersey - USA, 1997
- Bezerra, E. *Princípios de Análise e Projeto com a UML*, ed. Campus-Elsevier. 2003.