Projeto

Profa. Karin Becker
Engenharia de Software N
Instituto de Informática - UFRGS

Projeto OO

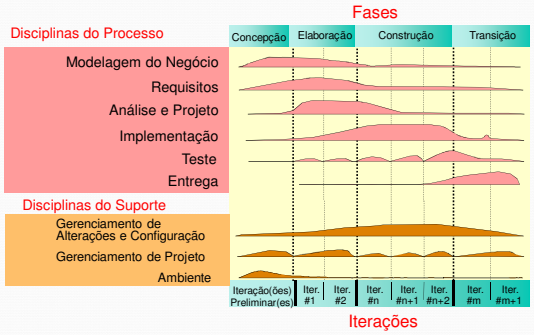
- Objetivo
  - Definir a arquitetura geral da aplicação
  - identificar as classes de software que serão implementadas
  - inclui definir atributos e métodos e posicioná-los nas classes
- Envolve
  - Distribuir responsabilidades como *métodos* das classes
    - usar *diagramas de colaboração*
  - introduzir classes de implementação (interface, banco de dados, utilitárias)
  - Aplicar padrões de projeto (*design patterns*)
    - Classes sem contrapartida no domínio, mas que resolvem problemas de projeto
  - Projeto de interface com o usuário, que capture eventos externos

Projeto OO : Algumas tarefas

- Definir Interface com Usuário
  - unidades de apresentação (telas, relatórios)
  - projeto da interação com o usuário (eventos externos, coordenação de eventos)
- Refinar Arquitetura do Sistema
  - Definir interfaces e dependências
  - Definir estrutura de comunicação
- Definir Diagramas de Interação
- Definir Diagramas de Classe de Projeto
  - Evoluir classes de análise para classes de software
  - Incluir classes de Persistência, de Interface, Utilitárias
- Definir Esquema de BD (Projeto de BD)

Processo Unificado : Análise e Projeto

Processo Unificado



Análise e Projeto: lembrando ...

- Objetivos básicos
  - Criar um projeto a partir dos requisitos identificados
  - Derivar uma arquitetura para o sistema
  - Adaptar o projeto para as limitações do ambiente de execução
- Centrado em Realização de Casos de Uso
- Pode ser dividido em dois modelos
  - Modelo de Análise (transitório)
  - Modelo de Projeto (permanente)

Modelo de Análise vs. Modelo de projeto

Modelo de Análise

- **TEMPORÁRIO**
- Define uma estrutura que funciona como **entrada fundamental ao processo de projeto**
- **Menos Esforço** (1/5 da disciplina)
- **Modelo conceitual** (uma abstração que evita detalhes de implementação)
- **Genérico** (mútiplas opções de projeto e tecnologia)
- **Menos formal**
- **Divisão de responsabilidades inicial**
- **Não foca muito em coordenação de interações**
- **Esboço do projeto** e de sua arquitetura

Modelo de Projeto

- Idealmente, **mantido** ao longo do ciclo de vida
- **Modelo "físico"** ("planta baixa") que funciona como entrada fundamental para implementação
- **Mais esforço** (4/5 da disciplina)
- **Específico** para uma implementação
- **Mais formal**
- **Foca muito em coordenação de interações**
- **Divisão de responsabilidades mais detalhada (refinamento)**
- **Detalha o projeto** e sua arquitetura
- **Define a estrutura do sistema**, mantendo ou não contrapartida no modelo de análise
- **Abstração** da implementação

Projeto : Objetivos

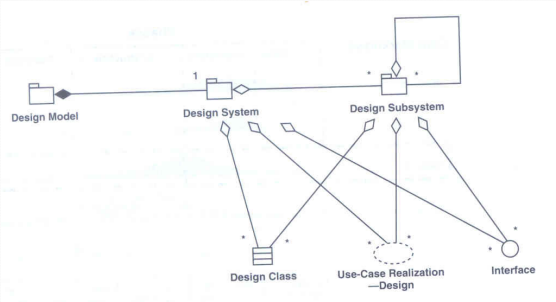
- Evoluir a estrutura derivada dos requisitos funcionais para uma solução de software
- Adquirir um entendimento aprofundado de tópicos ligados a requisitos não funcionais
  - Tecnologias, reuso, desempenho, etc
- Entrada para implementação
  - Subsistemas, interfaces, classes
  - Sincronização de subsistemas através de suas interfaces
  - Implementação concorrente por equipes
- Abstração unificada da implementação
  - Linguagem universal, gráfica
  - Rechear o código, preservando a estrutura

Projeto: Artefatos

- Modelo de Projeto
- Modelo de Implantação

Artefatos: Modelo de Projeto

Modelo de Projeto



Subsistema de Projeto

- Organiza os artefatos do modelo de projeto em unidades mais fáceis de gerenciar
  - Uma organização coesa da classes, realização de casos de uso, interfaces e outros subsistemas
  - Baixo acoplamento, alta coesão
- Representa uma separação de interesses

Aplicação (domínio)
Infraestrutura Tecnológica

  - Frequentemente os subsistemas da aplicação são mapeados a partir dos pacotes de serviço da análise
- Pode abstrair um software a ser reutilizado
  - componente, sistema legado, serviço de middleware, etc

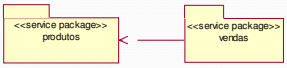
Interface

- Especificam operações providas por classes ou subsistemas

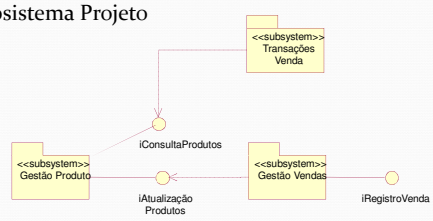
  - Subsistema: classes que realizam interface
  - Classe: operações que implementam interface
- Interfaces separam especificação funcional de implementações
  - Clientes dependem unicamente do que é exposto nas interfaces

Exemplo

- Subsistema Análise



- Subsistema Projeto



Classe de projeto

- Linguagem de especificação é a linguagem de programação alvo
- Definição da visibilidade de atributos e operações
- Tipos de atributos
- Assinatura detalhada das operações
- Realização das operações em pseudo-código
  - Postergar a consideração de alguns requisitos anotando-os como requisitos de implementação
- Relacionamentos são mapeados para mecanismos correspondentes na implementação
  - Especialização → Herança
  - Associação → variáveis
    - Determinar sentido de navegação
- Realização de interfaces de programação

Realização de Caso de Uso - Projeto

- Os casos de uso sugerem como os atores interagem com o sistema de software, o qual estamos interessados em criar.
  - um ator gera eventos reconhecíveis pelo sistema, solicitando como resposta alguma operação.
- Na **análise**, começa-se a considerar como estes eventos são gerenciados pelo sistema
  - Classes de análise
  - Esboço de colaborações
- No **projeto**, estas responsabilidades são refinadas
  - Classes de projeto
  - Operações
  - interações

Realização de Caso de Uso - Projeto

- Colaboração de objetos de projeto que realizam um caso de uso
  - Rastreados através da realização de caso de uso – análise
- Diagrama de classes:
  - operações, atributos e associações relevantes no contexto da realização do caso de uso
- Diagramas de interação
  - Seqüências de ações dos objetos/subsistemas resultantes de invocações do caso de uso
    - Realizar um fluxo de eventos considerando um cenário
    - Seqüência para detalhamento do fluxo de controle
    - Projeto alto nível – entre subsistemas
    - Projeto detalhado – entre objetos

Realização de Caso de Uso - Projeto

- Fluxo de eventos: documento textual que explica a relação entre múltiplos diagramas de interação
- Requisitos de implementação
  - Requisitos novos ou derivados do entendimento adquirido no projeto e que devem ser tratados em nível de implementação

Realização Caso de Uso - Projeto

Abstração de arquitetura de aplicações

- Decisões arquiteturais vão determinar a divisão de responsabilidades na aplicação
- No mínimo
  - Interface: como o usuário interage com o sistema
    - Apresentação, Controle da Interação com Usuário
    - Fornecimento de serviços, exibição de informação (p.ex windows ou HTML), tratamento de solicitações do usuario (clicks, teclas), requisicoes http, chamadas em linhas de comando
  - Semântica : o que a aplicação faz
    - Domínio:
      - Lógica de negócio que é o real propósito do sistema
      - Controle do processo
      - Objetos do Domínio
    - Fonte de Dados:
      - Comunicação com o banco de dados, sistemas de mensagens, gerenciadores de transacoes, outros pacotes

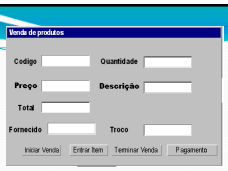
Realização de Caso de Uso - Projeto

- Sim, mas .... e eu começo por onde?
  - Entradas:
    - Modelo de análise – classes iniciais, colaborações, pacotes de serviços
    - Modelo de casos de uso : essenciais ...
  - Modelo de Interação com o usuário
    - Projeto de interface
    - A partir do projeto da interface, determinar o comportamento da aplicação
      - Casos de uso reais (ou concretos)
        - Conecta fortemente interface à interação com o sistema
        - Pouco usado
      - **Diagramas de Seqüência de Sistema (Larman)**
        - Concentra em eventos que podem vir da interface

Diagrama de Sequência do Sistema

- Não se preocupa com as classes de projeto da interface, somente o que acontece na aplicação para tratar um evento externo
- Ponto de partida
  - Eventos externos são gerados por atores interagindo com o sistema
  - Tipicamente, uma requisição para execução de alguma operação
  - Esta operação tem que ser tratada internamente pelo software
  - Quais os eventos, em que ordem acontecem, quando acontecem, qual o retorno?
  - **o sistema é visto como uma caixa-preta**
- para uma seqüência particular de **eventos** dentro de um **caso de uso** (**cenário**), modela
  - os **atores externos** que interagem diretamente com o sistema
  - os eventos que os atores geram e que devem ser tratados pelo sistema
  - O retorno ao ator externo
  - O tempo corre no sentido de cima para baixo, e a ordem dos eventos deve seguir sua ordem no caso de uso.
- Um diagrama de seqüência do sistema deve ser feito para a
  - seqüência típica de eventos do caso de uso
  - Sequencia alternativas

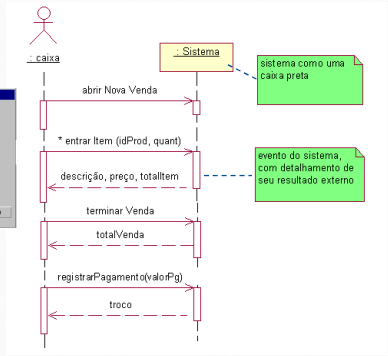
Exemplo

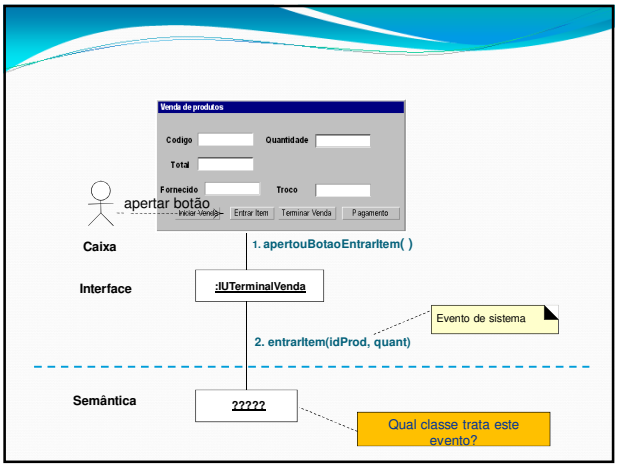


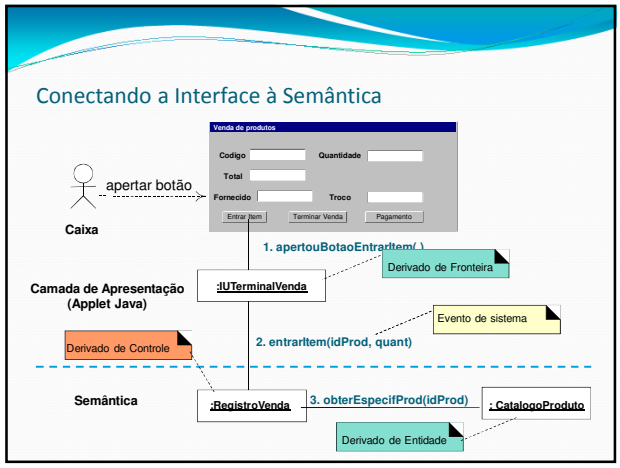
Identificação: UC1
Caso de uso: Registrar Venda
Atores: Caixa
Pré-Condições: o Caixa está logado no terminal, os produtos estão cadastrados
Pós-Condições: a venda é registrada, o estoque do produtos vendidos é atualizado, o pagamento é registrado e o recibo do cliente é impresso.
Seqüência Típica de Eventos (Fluxo Básico):

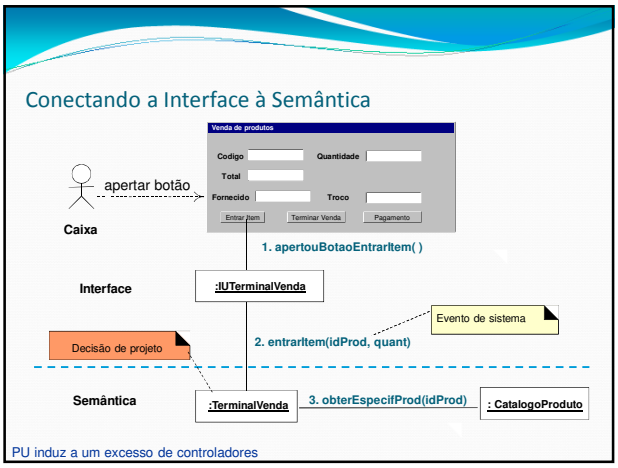
1. O Caixa inicia uma nova venda
2. Sistema registra nova venda
3. Para cada produto
   - O Caixa entra com a identificação e quantidade de itens do produto
   - O sistema registra a venda do produto e apresenta a sua descrição, preço e o valor total da venda de acordo com a quantidade de itens
4. O caixa informa o término da venda
5. O sistema apresenta o valor total da venda
6. O Caixa registra o pagamento recebido
7. O sistema registra a venda como completa, e imprime um recibo contendo dados da venda e do terminal onde esta foi efetuada.

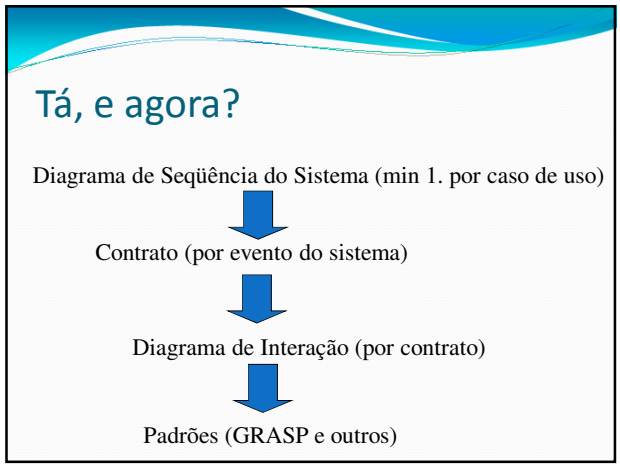
Registrar Venda




Venda de produtos
Codigo
Quantidade
Total
Fornecido
Troco
Iniciar Venda
Entrar Item
Terminar Venda
Pagamento
apertar botão
Caixa
1. apertouBotaoEntrarItem( )
Interface
:IUTerminalVenda
Evento de sistema
2. entrarItem(idProd, quant)
Semântica
?????
Qual classe trata este evento?


Conectando a Interface à Semântica
Venda de produtos
Codigo
Quantidade
Total
Fornecido
Troco
Entrar Item
Terminar Venda
Pagamento
apertar botão
Caixa
1. apertouBotaoEntrarItem( )
Derivado de Fronteira
Camada de Apresentação (Applet Java)
:IUTerminalVenda
Evento de sistema
Derivado de Controle
2. entrarItem(idProd, quant)
Semântica
:RegistroVenda
3. obterEspecifProd(idProd)
: CatalogoProduto
Derivado de Entidade


Conectando a Interface à Semântica
Venda de produtos
Codigo
Quantidade
Total
Fornecido
Troco
Entrar Item
Terminar Venda
Pagamento
apertar botão
Caixa
1. apertouBotaoEntrarItem( )
Interface
:IUTerminalVenda
Evento de sistema
Decisão de projeto
2. entrarItem(idProd, quant)
Semântica
:TerminalVenda
3. obterEspecifProd(idProd)
: CatalogoProduto
PU induz a um excesso de controladores


Tá, e agora?
Diagrama de Seqüência do Sistema (min 1. por caso de uso)
Contrato (por evento do sistema)
Diagrama de Interação (por contrato)
Padrões (GRASP e outros)

Contrato

- Associar cada operação do sistema com pré e pós condições
  - Pré-condições auxiliam a compreender classes envolvidas nas consultas e verificações a serem feitas
  - Pós-condições auxiliam a compreender as classes de projeto necessárias a partir das classes de análise
    - Criação de objetos / remoção
    - Criação de associações / remoção
    - Alteração do estado (atributos) de objetos
  - Larman sugere que se documente os contratos ...
    - Avaliar se vale a pena o esforço
    - O que agrega valor é **refletir** sobre objetos necessários

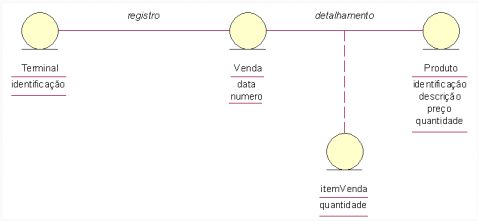
Exemplo: Registrar Venda

Identificação: UC1
Caso de uso: registrar Venda
Atores: Caixa
Pré-Condições: o Caixa está logado no terminal, os produtos estão cadastrados
Pós-Condições: a venda é registrada, o estoque do produtos vendidos é atualizado, o pagamento é registrado e o recibo do cliente é impresso.
Seqüência Típica de Eventos (Fluxo Básico):

1. O Caixa inicia uma nova venda
2. Sistema registra nova venda
3. Para cada produto
   - O Caixa entra com a identificação e quantidade de itens do produto
   - O sistema registra a venda do produto e apresenta a sua descrição, preço e o valor total da venda de acordo com a quantidade de itens
4. O caixa informa o término da venda
5. O sistema apresenta o valor total da venda
6. O Caixa registra o pagamento recebido
7. O sistema registra a venda como completa, e imprime um recibo contendo dados da venda e do terminal onde esta foi efetuada.

Modelo de Análise

Somente Classes
Entidade (simplificado)

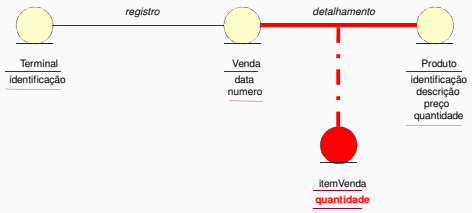


Exemplo Contrato : entrarItem

Operação: entrarItem (idProd, quantidade)

- Referências Cruzadas: Caso de Uso: Vender Itens
- Pré-condições: O idProd é conhecido do sistema
- Pós-condições:
  - Uma associação entre Venda e Produto foi criada (criação de associação)
  - Uma instância de ItemVenda foi criada (criação de instância)
  - Novo ItemVenda foi associado à Venda corrente (criação de associação)
  - ItemVenda.quantidade (ItemVenda) recebeu o valor de quantidade (modificação de atributo)
  - Novo ItemVenda foi associado a um Produto, baseado numa correspondência com o idProd (criação de associação)

Pós-condições: entrarItem



Padrões de Projeto

- Padrão: uma solução comum para um problema comum em um determinado contexto
- General Responsability Assignment Software Patterns – GRASP (Craig Larman)
  - Princípios fundamentais (e básicos) de distribuição de responsabilidades
  - Muiiiiiiito básicos ...
- Padrões de Projeto (GOF – Gamma et al.)
  - Princípios avançados para distribuição de responsabilidades voltados em particular à extensão e reuso

Padrões GRASP

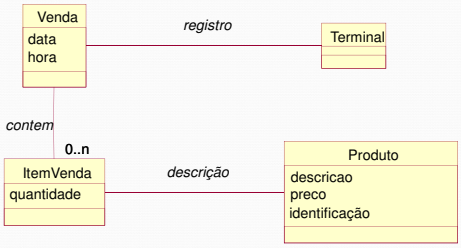
- 5 princípios fundamentais para distribuição de responsabilidades nas classes
  - Expert (especialista)
  - Creator (Criador)
  - High Coesion (Alta coesão)
  - Low Coupling (Baixo Acoplamento)
  - Controller (Controlador)
- Responsabilidades: um contrato ou obrigação de um objeto
  - Responsabilidade de fazer :
    - fazer algo ele próprio
    - iniciar ações em outros objetos
    - controlar e coordenar as atividades em outros objetos
  - Responsabilidades de conhecer :
    - conhecer dados privados encapsulados
    - conhecer objetos relacionados
    - conhecer coisas que ele pode derivar e calcular

GRASP: Padrão Especialista

- Problema: qual o princípio mais básico para atribuir responsabilidades a objetos?
- Solução: atribuir a responsabilidade ao especialista da informação, i.e. a classe que tem a informação necessária para honrar a responsabilidade
  - defina claramente qual a responsabilidade antes de tentar atribuí-la
- Exemplo: quem deve conhecer o total de uma venda?

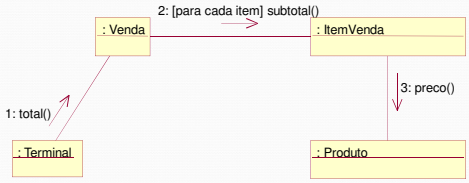
Exemplo: Padrão Especialista

- **quem deve conhecer o total de uma venda?**
  - especialistas candidatos



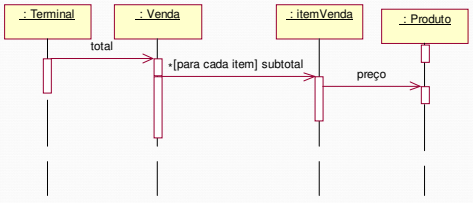
Exemplo: Padrão Especialista

- **quem deve conhecer o total de uma venda?**



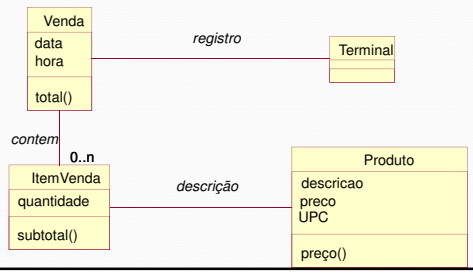
Exemplo: Padrão Especialista

- **quem deve conhecer o total de uma venda?**



Exemplo: Padrão Especialista

- **quem deve conhecer o total de uma venda?**
  - Distribuição de responsabilidades aos especialistas



Padrão Especialista

- Atenção: um pequeno problema com este padrão ...
  - **e a atribuição da responsabilidade de conhecer a informação (caracterização de especialista), foi feita como?**
  - "as responsabilidades relacionadas a "conhecer" são freqüentemente dedutíveis do modelo de análise, por causa dos atributos e associações que ele ilustra"
  - **Não subestimar dificuldade desta tarefa !!!!**
    - **Atributos : mais ou menos fácil**
    - **Associações : Decidir sobre navegabilidade e visibilidade de associações não é sempre trivial!!**

GRASP: criador

- problema: quem deve ser o responsável pela criação de uma nova instância de alguma classe?
- Solução
  - atribua à classe B a responsabilidade de criar uma instância da classe A nas seguintes situações
    - B agrega/contém objetos A
    - B registra instâncias de A
    - B usa de maneira muito próxima objetos de A
    - B tem os dados de inicialização para criação de A

GRASP: Controlador

- Problema: quem é responsável por tratar eventos do sistema?
  - Evento de sistema é um evento gerado por um ator externo.
  - Um controlador é um objeto responsável por tratar um evento do sistema
  - Um controlador define o método para **operação de sistema**.
- Solução
  - Controlador de Caso de Uso:
    - Usar a mesma classe controladora para todos os eventos de sistema do mesmo caso de uso
  - Controlador fachada:
    - Usar uma classe representando o sistema como um todo
    - Adequado somente quando existem poucos eventos do sistema
  - Combinações (fachada de casos de uso)

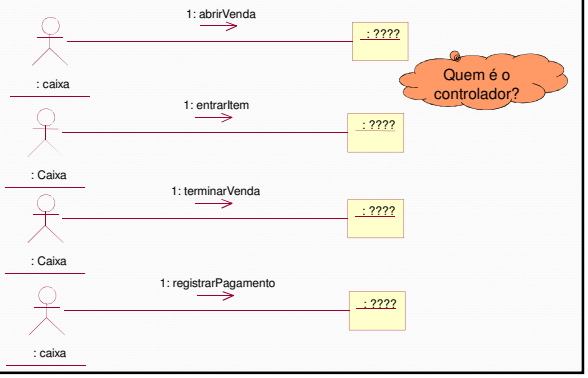
GRASP: Baixo Acoplamento

- Problema: como dar suporte à baixa dependência e aumentar a reutilização?
- Solução
  - atribuir uma responsabilidade de modo que o acoplamento permaneça fraco (pouca dependência de outras classes)
    - reduzir o número de objetos com o qual um objeto se comunica
    - simplificar a interface através da qual um objeto se comunica (operações de maior granularidade)

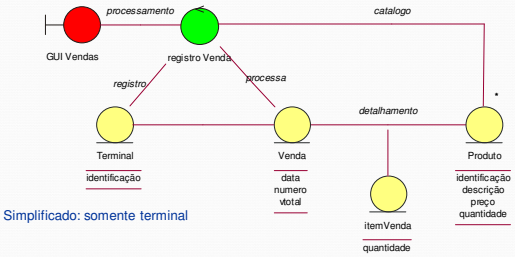
GRASP: Alta Coesão

- Problema: como manter a complexidade sob controle?
- Solução
  - atribuir uma responsabilidade de modo que o a coesão permaneça alta
    - cada objeto faz "uma coisa" só
    - objetos devem ter poucas responsabilidades

Realização de Caso de Uso - GRASP



Registrar Venda: modelo de análise



Simplificado: somente terminal

Processo unificado induz a um controlador por caso de uso no modelo de análise, mas nem sempre esta é uma boa decisão de projeto

Projeto: PADRÃO CONTROLADOR

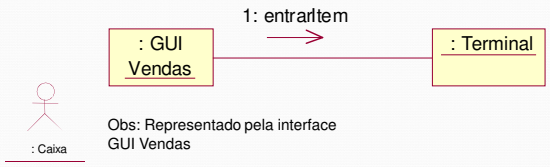
Contrato : entrarItem

Operação: entrarItem (idProd, quantidade)

- Referências Cruzadas: Caso de Uso: Vender Itens
- Pré-condições: O idProd é conhecido do sistema
- Pós-condições:
  - Uma associação entre Venda e Produto foi criada (criação de associação)
  - Uma instância de ItemVenda foi criada (criação de instância)
  - Novo ItemVenda foi associado à Venda corrente (criação de associação)
  - ItemVenda.quantidade (ItemVenda) recebeu o valor de quantidade (modificação de atributo)
  - Novo ItemVenda foi associado a um Produto, baseado numa correspondência com o idProd (criação de associação)

Exemplo entrarItem(idProd, qtd)

- Candidatos a Controlador
  - caso de uso: registroVenda (modelo de análise)
  - fachada: Terminal, crtlSuperMercado
- decisão:
  - poucos eventos no caso de uso e na aplicação como um todo
  - Terminal é o controlador

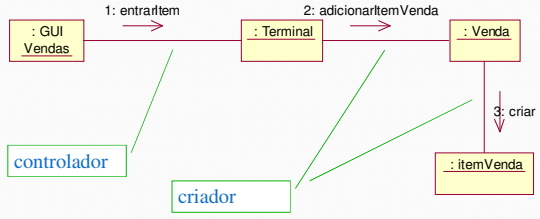


Obs: Representado pela interface GUI Vendas

Exemplo entrarItem(idProd, qtd)

- criar item de Venda
  - Venda gerencia a coleção de linhas de item (é responsável por registrar a linha, usa de maneira muito próxima a linha de item)

Exemplo entrarItem(upc, qtd)

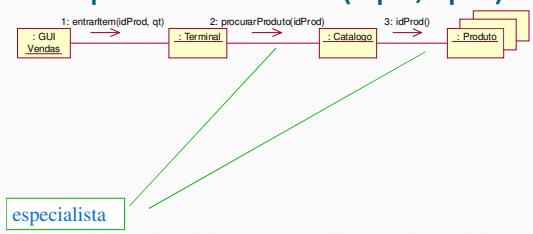


Exemplo entrarItem(idProd, qtd)

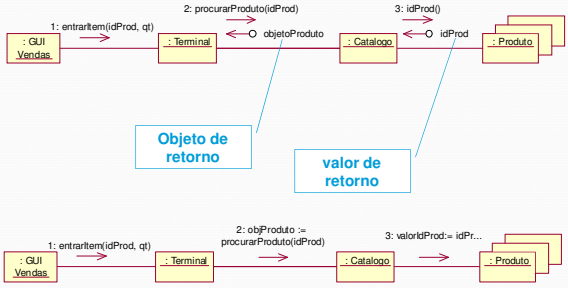
- Encontrar especificação de produto (baseado em idProd)

- quem é responsável por conhecer a especificação de UM produto?
  - Produto
- quem é responsável por conhecer uma especificação baseada em uma correspondência de idProd (coleção de especificações)?
  - Associação catalogo (criar classe catalogoProdutos)
- quem é responsável por conhecer o catálogo?
  - Terminal (o controlador adotado)

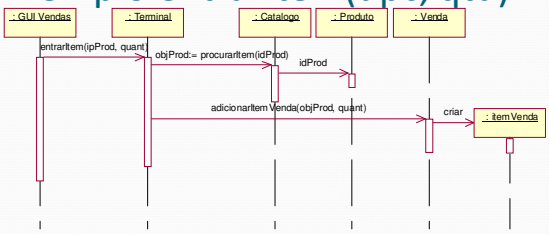
Exemplo entrarItem(upc, qtd)



Exemplo entrarItem(upc, qtd)



Exemplo entrarItem(upc, qtd)



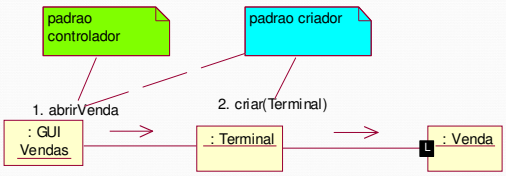
Realização Caso de Uso Comprar Itens: contrato abrirVenda

- Operação: abrirVenda (Terminal)
- Referências Cruzadas: Caso de Uso: Vender Itens
- Pré-condições:
- Pós-condições:
  - Uma instância de Venda foi criada (criação de instância)
  - Nova Venda foi associado ao Terminal corrente (formada uma associação)

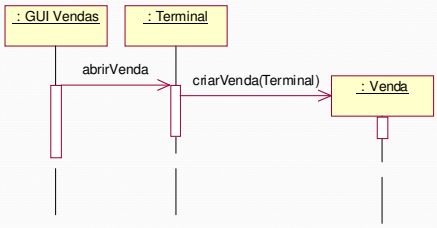
Exemplo: abrirVenda

• Quem é o controlador?
(Fachada → Terminal)

• Quem é o responsavel por criar Venda
•Terminal é responsável por registrá-la



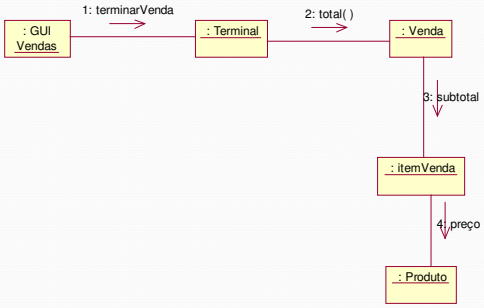
Exemplo: abrirVenda



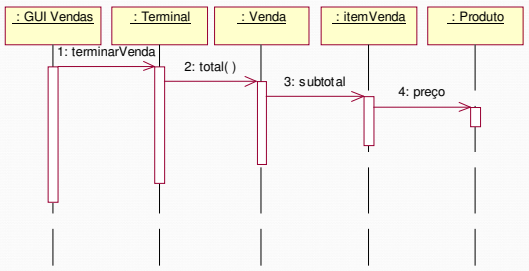
Realização Caso de Uso Comprar Itens: contrato Terminar Venda

- Operação: terminarVenda ()
  - Referências Cruzadas: Caso de Uso: Vender Itens
  - Pré-condições:
  - Pós-condições:
    - Venda.estado é atualizado para concluída
    - Venda.Total é calculado e atualizado
- Quem é o controlador?
  - Terminal
- Quem é o especialista?
  - Produto: preço
  - itemVenda: quantidade
  - Venda = todos os itens

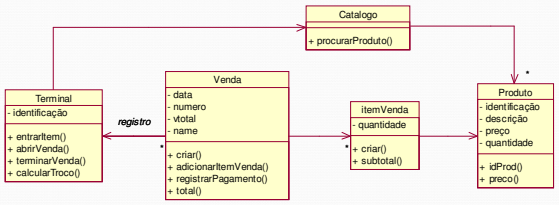
Exemplo: Terminar Venda



Exemplo: Terminar Venda



Realização de Caso de Uso – Projeto
Diagrama de Classes resultante



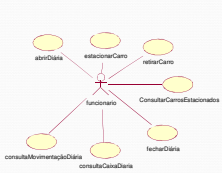
Que mais?
- Adicionar tipo aos atributos
- Adicionar argumentos (entrada/saida) nas operações
- classes persistência, utilitárias

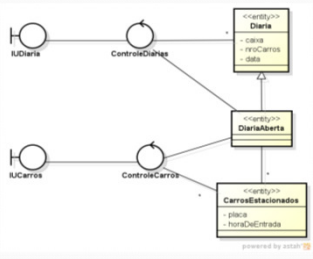
Para saber mais ...

- Leitura recomendada
  - Craig Larman. Utilizando UML e Padrões. Bookman. (2d edição mais enxuta, 3ed mais completa)
    - Diagramas de Sequência : cap 10
    - Contratos: cap 11
    - GRASP: 17
- Leitura sugerida
  - Gamma, E. et al. Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. Bookman.

Exercício

- Um projeto muito simples : Estacionamento





Prática

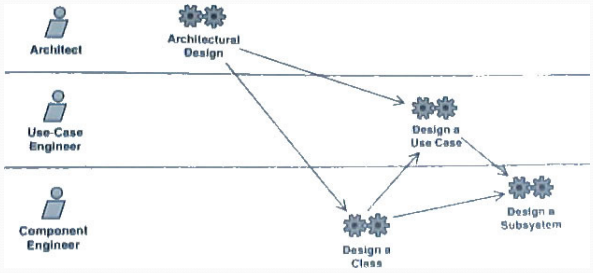
- Modelo de Projeto para estudo de caso Biblioteca
  - Identificar subsistemas e suas dependências
  - Refinar interfaces
  - Realizar os casos de uso desenvolvidos na análise
    - Classes de projeto
    - Colaborações
      - Seqüência ou colaboração
  - Refinar arquitetura

Atividades

- Elaborar Diagrama de Classes de Projeto
  - Identificar classes adicionais, atributos específicos
  - Identificar associações e agregações
    - Sentido de navegação
  - Identificar generalizações
  - Descrever os métodos
    - Progressivamente com todos os detalhes necessários
  - Descrever os estados
  - Definir os requisitos especiais
  - Identificar interfaces
- Pacotes de Projeto (Subsistemas)

Análise e Projeto: Atividades de Atores

Atividades e Atores



Projeto: Atividades

- Análise arquitetural (arquiteto)
  - Identificar susbsistemas e interfaces
    - Pacote de análise → subsistema
    - Refinar funcionalidade compartilhada, encapsulamento de software legado, etc
    - Refinar dependências e camadas
    - Definir interfaces que reflitam as dependências
    - Identificar operações das interfaces
  - Identificar classes de projeto significativas
  - Identificar mecanismos de projeto genérico
    - Persistência, distribuição (transparente), transações, detecção de falhas, etc
  - Considerar estilos arquiteturais, padrões arquiteturais, reutilização de componentes
  - Identificar nodos e a configuração de rede
    - Nodos: número e capacidade de processamento
    - Conexões: protocolos, largura de banda, qualidade, etc
    - Tolerância a falhas: replicação, backup, fail-over, etc

Projeto : Atividades

- Projeto de Caso de Uso (engenheiro de casos de uso)
  - Realização dos Casos de Uso - Projeto
  - Identificar as classes de projeto participantes da realização
    - Derivadas da realização de caso de uso – análise
    - Derivadas dos requisitos especiais do caso de uso
    - Descobertas durante o projeto
  - Descrever as interações entre os objetos de projeto
    - Iniciado por um evento de sistema (uma mensagem do ator)
    - Mensagens devem ser (ou se tornar) operações em classes de projeto
    - Usar notas ou complementar com descrições textuais do fluxo de eventos

Projeto : Atividades

- Projeto de Caso de Uso (engenheiro de casos de uso)
  - Identificar os subsistemas e interfaces participantes
    - Mostrar as interações em nível de subsistema
    - Diferentes níveis de granularidade
      - Objetos
      - Susbsistemas
  - Capturar requisitos de implementação
  - Utilizar padrões de projeto e boas práticas

Projeto : Atividades

- Projeto de Classes de Projeto (engenheiro de componentes)
  - Elaborar Diagrama de Classes de Projeto
  - Identificar classes adicionais, atributos específicos
  - Identificar associações e agregações
    - Sentido de navegação
  - Identificar generalizações
  - Descrever os métodos
    - Progressivamente com todos os detalhes necessários
  - Descrever os estados
  - Definir os requisitos especiais
  - Identificar interfaces

Projeto : Atividades

- Projeto de Subsistema (engenheiro de componentes)
  - Assegurar que o subsistema
    - Seja o mais independente possível dos demais subsistemas
    - Provê as interfaces apropriadas
    - Cumpre o seu propósito (realização de suas interfaces)
  - Depender de interfaces, não dos componentes internos dos subsistemas
  - Realocar classes a outros subsistemas para reduzir dependências
  - Definir colaborações para mostrar como um subsistema realiza suas interfaces
    - Componentes internos do subsistema