Lição 4

Engenharia de Projetos – Parte 1

Objetivos

Ao final desta lição, o estudante será capaz de:

- Aprender os conceitos e dinâmicas da engenharia de projetos
- Aprender a como desenhar a arquitetura de um software
- Aprender como desenvolver o modelo de dados
- Aprender a como desenhar a interface, particularmente, as telas e diálogos
- Aprender a como projetar os componentes de um software
- Aprender a como utilizar os requisitos de rastreamento
- Aprender sobre as Métricas de Projeto

Engenharia de Projetos

- Possui o foco na criação de uma representação ou modelo
- Envolve um processo iterativo de refinamento do mais alto nível de abstração até o mais baixo

Dicas de Qualidade para o Projeto

- Possuir uma arquitetura reconhecível
- Ser modular
- Fazer uso de uma notação que traga significados
- Nas estruturas de dados deverão guiar para o apropriado design das classes que foram derivadas de padrões de dados conhecidos
- Dos componentes deverá possuir características funcionais independentes
- Possuir interfaces que reduzam a complexidade das ligações entre os componentes e o ambiente
- Derivado do repetitivo uso de um método para obter informações durante a fase de engenharia de requisitos

Conceitos de Projeto

- Abstração
- Modularidade
- Refinamento
- Refatoramento

Abstração

- Dois tipos de Abstração
  - Abstração de Dados
    - Se refere a uma nomeada coleção de dados que descrevem a informação requisitada pelo sistema
  - Abstração Procedural
    - Se refere as seqüências de comandos ou instruções que possuem ações especificas limitadas

Modularidade

- É a característica em um Software que permite que ele seja desenvolvido e mantido pela sua decomposição em pequenas partes de trabalho denominadas **módulos**

- Ele nos leva ao **ocultamento de informações**, que significa que detalhes serão ocultos de outras classes e módulos

- Ele incentiva a **Independência Funcional** que é a característica de um modulo ou uma classe para endereçar uma função especifica como o definido em seus requisitos

Acoplamento

- Grau de interconexão entre os objetos e pelo grau de interação que eles possuem com outros
  - Acoplamento Interativo
  - Acoplamento de Herança

Coesão

- Medida na qual um elemento contribui para um único propósito
  - Coesão Operacional
  - Coesão de Classe
  - Coesão de Especialização

Refinamento

- Conhecido como o processo de elaboração
- Descobre detalhes a medida que o desenvolvimento progride

Modelo de Projeto

- Projeto de Arquitetura
- Projeto de Dado
- Projeto de Interface
- Projeto de Componentes
- Projeto de Implementação

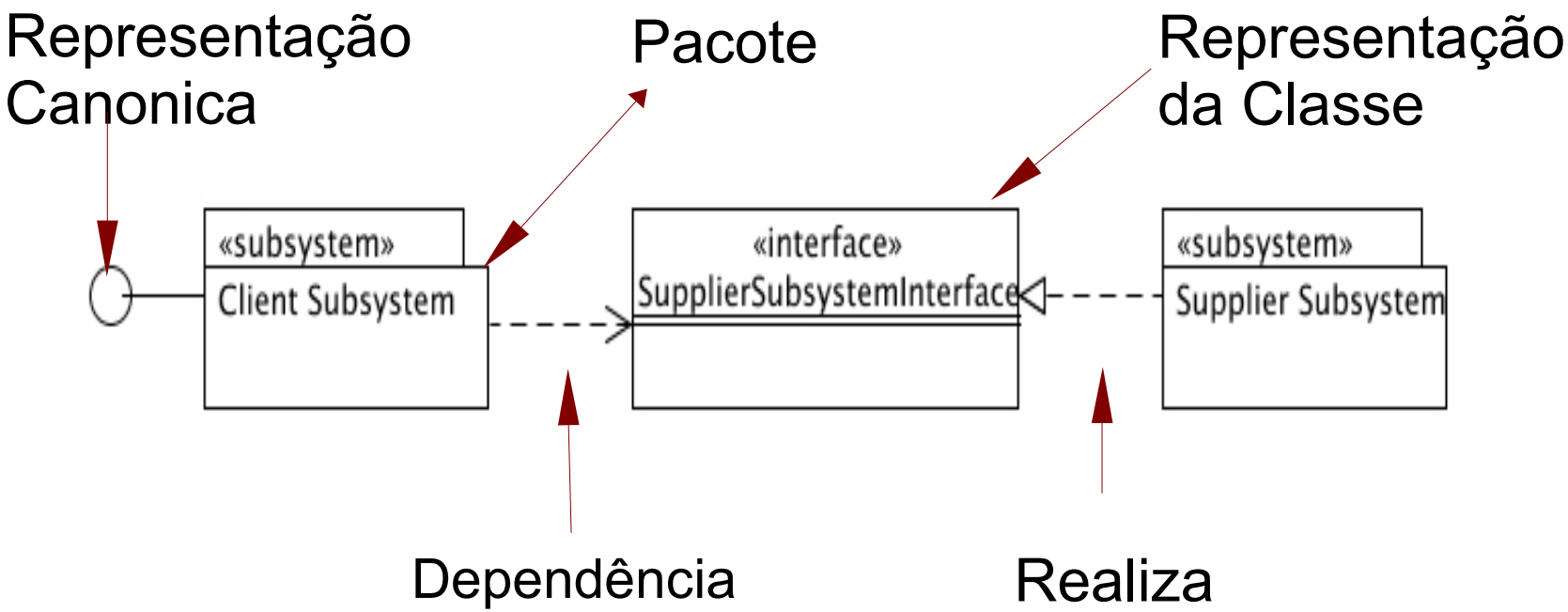
Arquitetura de Software

- Visão geral da estrutura do software e os caminhos para que a estrutura permita a integridade para um sistema
- Os componentes do Software estruturados em camadas
- Decisão de como o software será construído e como irá controlar o desenvolvimento incremental do sistema

Diagrama de Pacote

- Mostra a quebra de grandes sistemas em pequenos grupos lógicos
- Mostra os grupos de classes e as dependências entre elas

Diagrama de Pacote

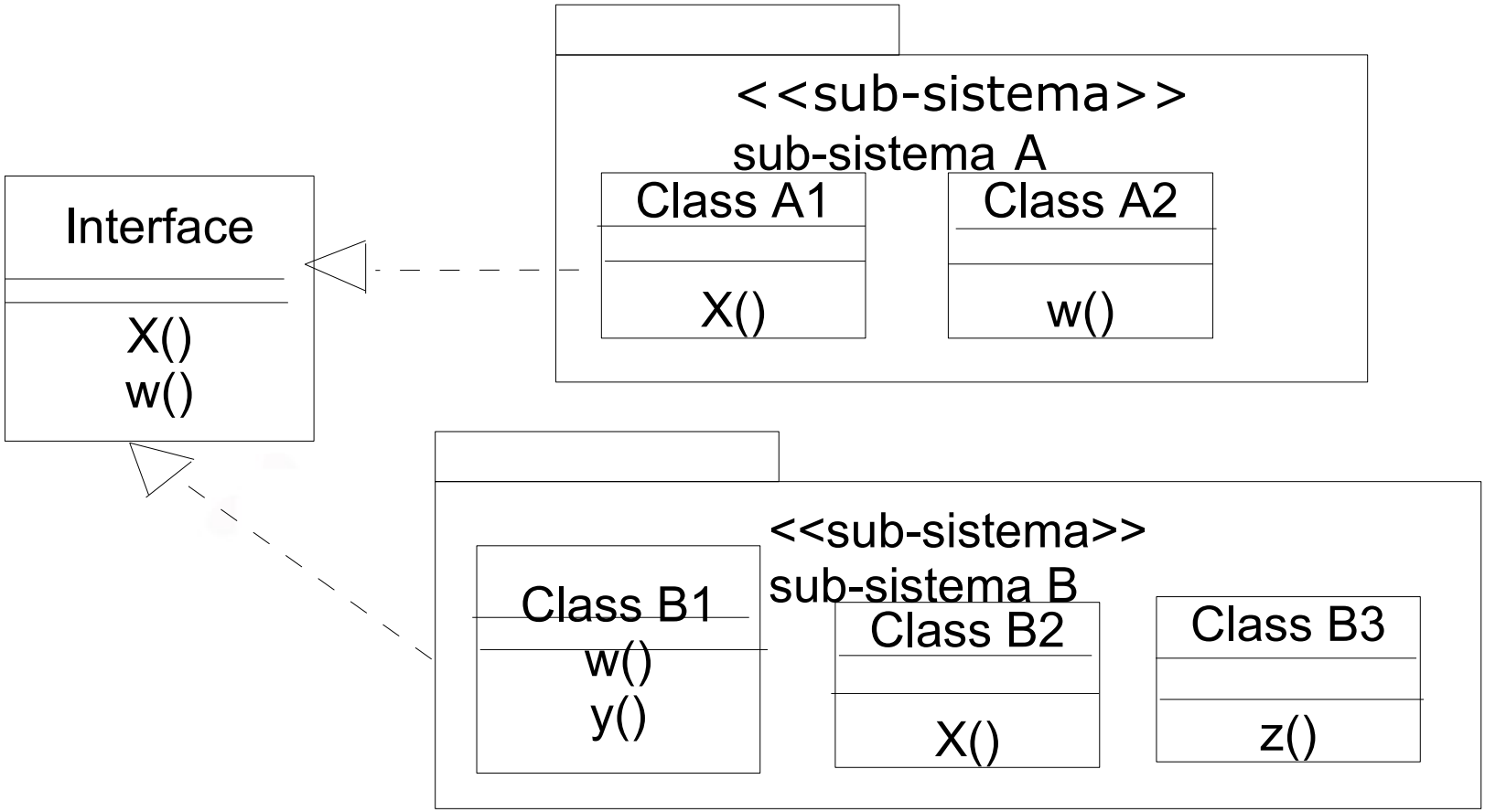


Sub-Sistema e Interface

- Sub-Sistema
  - Combinação de um pacote(pode conter outros elementos) e uma classe(ela interage com outros elementos do modelo)
  - Utilizado para particionar o sistema em partes que podem ser independentemente desenvolvidos, configurados e entregues

- Interface
  - Define uma série de operações que são realizados pelo sub-sistema
  - Permite uma separação entre a declaração de comportamento e sua realização

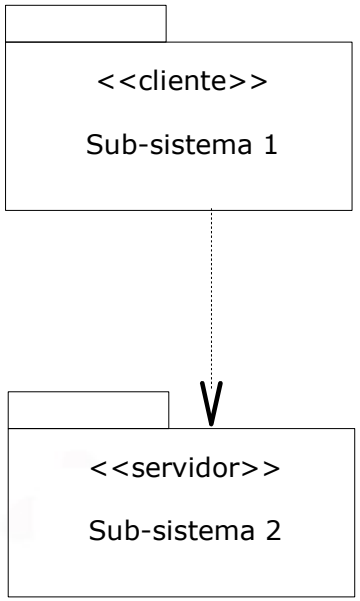
Sub-Sistema e Interface



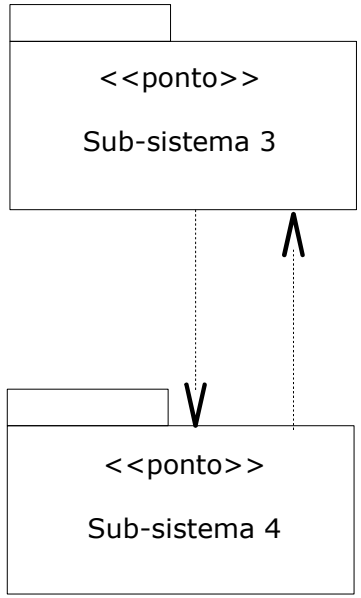
Estilos de Comunicação de Sub-Sistemas

- Comunicação Cliente-Servidor
  - O Sub-sistema Cliente necessita conhecer a interface para o Sub-sistema Servidor
  - Comunicação de único caminho
  - Fácil de implementar e dar manutenção
- Comunicação Ponto-a-Ponto
  - Cada sub-sistema conhece a interface do outro sub-sistema
  - Uma comunicação de caminho duplo

Estilos de Comunicação de Sub-Sistemas



O sub-sistema servidor não depende do sub-sistema cliente. Mudanças na interface de cliente não o afetam.

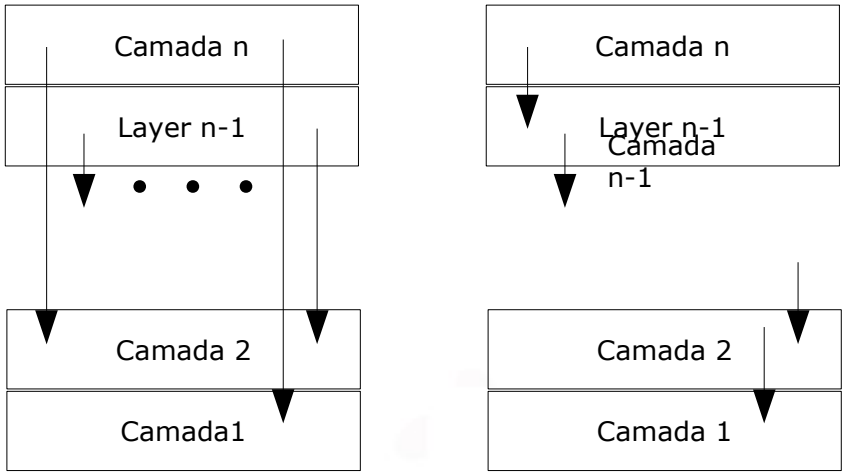


Cada sub-sistema do ponto depende do outro. Eles são afetados por mudanças em cada uma das outras interfaces.

Modos de Dividir o Software em Sub-Sistemas

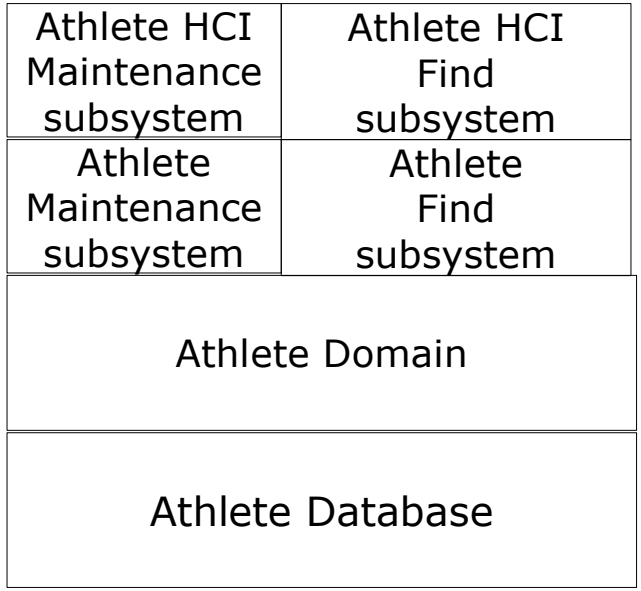
- Por Camadas
  - Focado em sub-sistemas que são representados por diferentes leveis de abstração ou camadas de serviço
- Por Particionamento
  - Focado em diferentes aspectos na funcionalidade do sistema como um todo.

Estrutura Geral da Arquitetura por Camadas

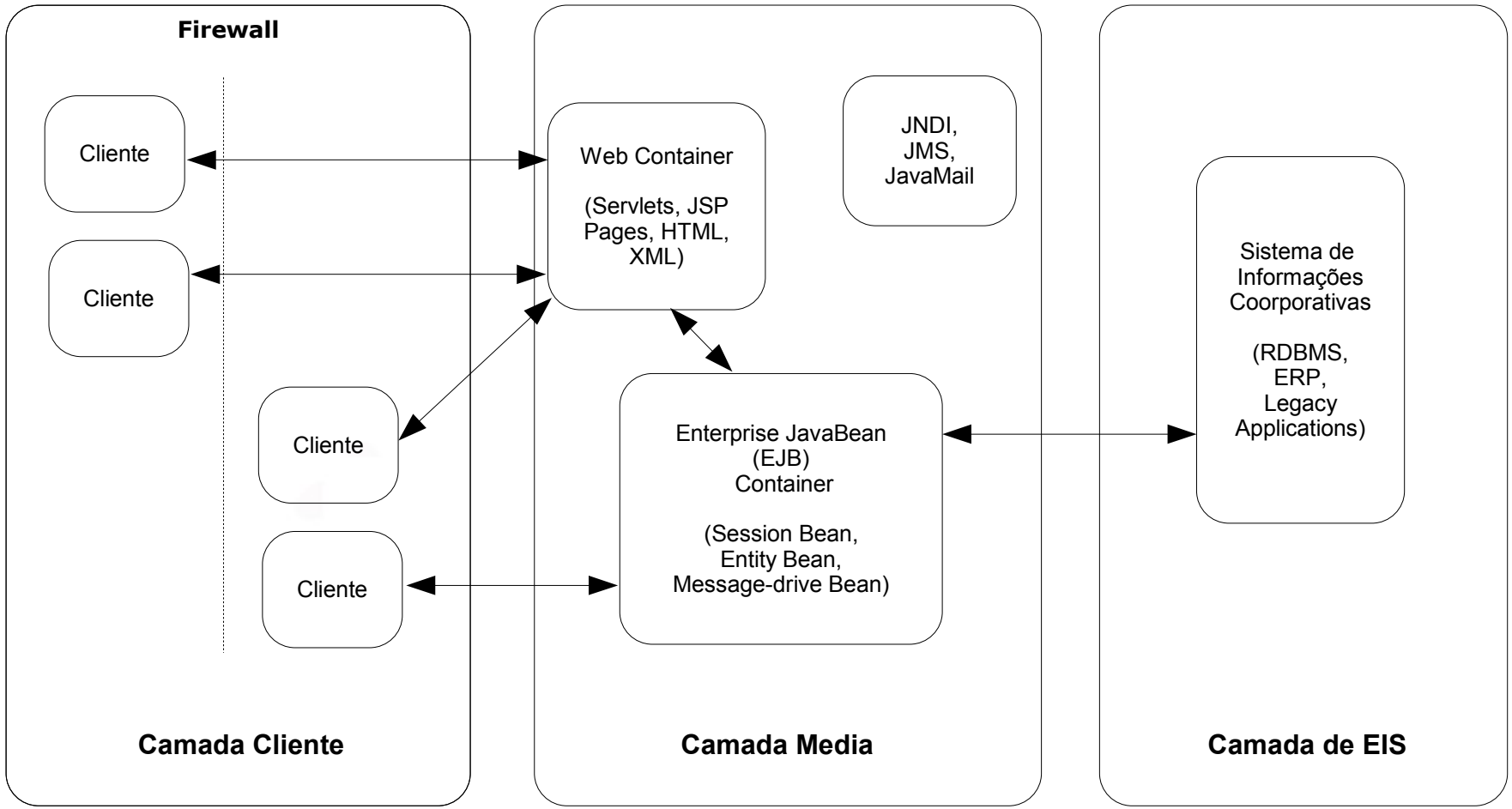


- Arquitetura Aberta
  - Os Sub-sistemas podem requisitar serviços de qualquer camada abaixo deles
- Arquitetura Fechada
  - Os sub-sistemas podem somente requisitar serviços da camadas diretamente abaixo dele
  - Sub-sistemas não são permitidos para pular outros sub-sistemas

Exemplo por Camada e por Partição



Amostra de Arquitetura por Camadas na Plataforma J2EE



Desenvolvendo o Projeto para a Arquitetura

- Validar a análise das classes
- Agrupar as classes relacionadas em pacotes
- Identificar classes e subsistemas do projeto
- Definir a interface do sub-sistema
- Organizar os sub-sistemas e classes em camadas para a arquitetura

Passo 1: Validação da Análise das Classes

- Assegura que os atributos e responsabilidades serão distribuídos nas classes
- Assegura que uma classe defina somente uma abstração lógica

Passo 2: Agrupando Classes relacionadas em Pacotes

- Organize as classe juntas em pacotes que podem ser baseadas em configuração de unidade, alocação de recursos, grupos de tipos de usuários e representação do produto existente e de serviços que o sistema faz uso

- Também identifica a visibilidade dos pacotes e seus acoplamentos

Dicas para o Empacotamento

- Limite para o empacotamento de classes
  - Se as interfaces do sistema são parecidas
  - Se nenhuma grande modificação nas interfaces for planejada
  - Classes que não são relacionadas funcionalmente
  - Classes que são relacionadas em um serviço opcional

Dicas para o Empacotamento

- Empacotando Classes com Funcionalidades Relacionadas
  - Se uma mudança numa classe modificar outra
  - Se uma classe for removida de um pacote e isto causar um grande impacto
  - Quando duas classes são funcionalmente relacionadas
  - Duas classes relacionadas a atores diferentes
  - Classes mandatórias e opcionais

Acoplamento de Pacotes

- Como as dependências são definidas entre os pacotes
  - Não devem estar em acoplamentos cruzados
  - De camadas mais baixas não devem ser dependentes dos pacotes nas camadas mais altas
  - Não deverão pular as camadas
  - Não deverão ser dependentes do sub-sistema

Passo 3: Identificar classes e Subsistemas do Projeto

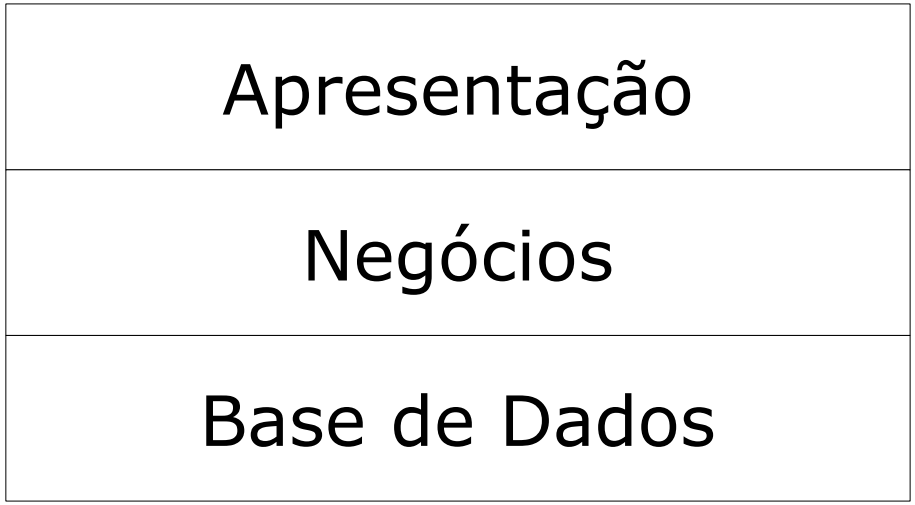
- Determina se a análise das classes são realmente classes ou subsistemas.
- Ele decide:
  - Quais classes analisadas são realmente classes e quais não são
  - Quais classes analisadas são subsistemas
  - Quais componentes existem; quais componentes precisam ser desenhados e implementados

Passo 4: Definir a Interface do Sub-Sistema

- Define a interface do sub-sistema
- Interface
  - È um grupo de operações visíveis de um sub-sistema
  - Não possui uma estrutura interna, nem detalhes da implementação dos atributos e operações
  - Define um grupo de operações que deve ser realizado pelo sub-sistema

Passo 5: Camadas do Sub-Sistema e Classes

- Os elementos do design necessitam ser alocadas em camadas especificas na arquitetura
- Em geral, a maioria das classes limites tende a aparecer no topo das camadas; classes de controle tendem a aparecer no meio e classes de entidades tendem a aparecer no fundo



Dicas para Arquitetura em Camadas

- Considerando a Visibilidade
- Considerando a Volatilidade
- Número de Camadas

Padrões de Projeto

- São soluções comprovadas para problemas recorrentes
- São soluções que podem ser reutilizadas para problemas em comum
- Elas descrevem uma forma como o problema é solucionado;eles não são soluções para os problemas
- Eles não são FRAMEWORKS

Frameworks

- São softwares parcialmente completos que podem servir para um tipo de aplicação
- Aplicações são construídas a partir de *frameworks* completando os elementos não terminados e adicionando elementos específicos
- Classes são especializadas e sua implementação é criada

Composite View

- Contexto
  - Fazer as visões gerenciáveis definindo modelos para tratar páginas comuns
- Problema
  - Dificuldades para modificar e gerenciar a disposição das múltiplas visões geram códigos duplicados
- Solução
  - Utilize este padrão se a visão for composta por múltiplas visões atômicas
  - Esta solução fornece a criação de uma visão composta através da inclusão e substituição de componentes dinâmicos

Front Controller

- Contexto
  - Fornece um controlador central para o gerenciamento de requisições. Recebe todas as requisições dos clientes, encaminha cada requisição para seu manuseador e apresenta uma resposta ao cliente
- Problema
  - Se não existirem controladores centralizados, problemas podem ocorrer
- Solução
  - Um controlador é definido para servir como ponto inicial de contato para o manuseio das requisições.
  - Os serviços podem incluir(autorização e autenticação) , delegando processos de negócios, gerenciamento de próximas visões, etc.

Data Access Object

- Contexto
  - Separa a interface do cliente do mecanismo de acesso aos dados
  - Permite a modificação dos mecanismos de acesso aos dados independentes do código que utiliza os dados
- Problema
  - Os dados estarão sendo recebidos de diferentes mecanismos de armazenamento tais como bancos de dados relacionais, mainframes, sistemas legados, serviços externos
- Solução
  - Utilize o DAO para abstrair e encapsular todo o acesso a fonte dos dados
  - Isto implementa os mecanismos de acesso aos dados

Abstract Factory

- Método estratégico de Fábrica
  - Ela é utilizada se a camada de armazenamento não for subjetiva a mudar de uma implementação para a outra

- Padrão *Abstract Factory*
  - Ela é utilizada se a camada de armazenamento for modificada de uma implementação para outra

Model-View-Controller

- Contexto
  - A aplicação apresenta seu conteúdo para os usuários pode meio de numerosas páginas contendo dados
- Problema
  - Se existir a necessidade de suporta a múltiplos tipos de usuários com múltiplos tipos de interface
- Solução
  - É largamente utilizado para aplicações interativas
  - Divide a funcionalidade entre os objetos envolvidos na manutenção e apresentação dos dados para minimizar o grau de acoplamento entres os objetos

Final da Parte 1

- Continua...

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**