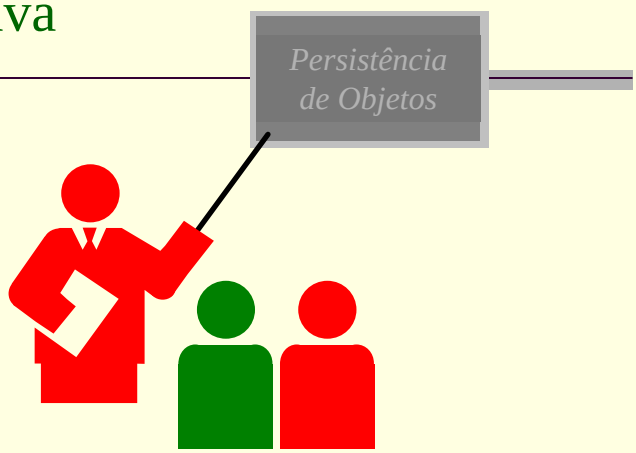
Módulo III
Camada de Persistência

Prof. Ismael H F Santos

Ementa

- Modulo III – Camada de Persistência
  - Persistência de Objetos
  - Mecanismo de Persistência
  - Padrões de Persistência

POO-Java



Objetos Transientes x Persistentes

- Os objetos de um sistema podem ser classificados em persistentes e transientes.
  - *Objetos transientes:* existem somente na memória principal.
    - Objetos de controle e objetos de fronteira.
  - *Objetos persistentes*: têm uma existência que perdura durante várias execuções do sistema.
    - Precisam ser armazenados quando uma execução termina, e restaurados quando uma outra execução é iniciada.
    - Tipicamente objetos de entidade.

Persistência de Objetos

- Por que não Bancos de Dados OO?
  - Tipos de dados e modelo de objetos diferentes da linguagem de programação
  - Estruturas de dados otimizadas para disco são diferentes as estruturas otimizadas para memória
  - Bancos OO ainda não são tão maduros quanto os relacionais em respeito a recoverabilidade, segurança, performance e distribuição

Persistência de Objetos

- Porque Bancos de Dados Relacionais ?
  - Têm se mostrado bastante eficiêntes, o que levou a uma boa aceitação do mercado
  - Algumas aplicações utilizam-se de Stored Procedures e Triggers para execução de regras de negócio
  - Atualmente grande parte da Aplicações utilizam e ou integram-se por meio de um Banco de dados Relacional

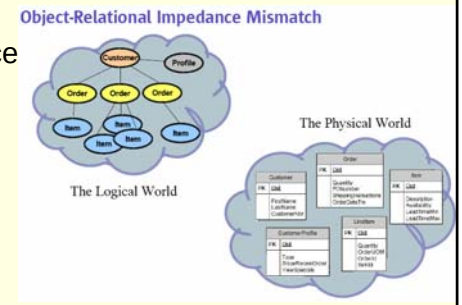
POO-Java

Componentes de um Mecanismo de Persistência

- **Mapeamento OO/Relacional**
- **Linguagem de Consulta**
- **API de acesso**

Componentes de um Mecanismo de Persistência

- **Mapeamento OO/Relacional**
  - Um modelo OO é semanticamente mais rico do que um modelo Relacional
  - Vários modelos Relacionais são possíveis para um mesmo modelo OO
  - Objetivos de performance e confiabilidade podem levar a mapeamentos "degenerados", semelhantes a bancos relacionais não normalizados



Mapeamento OO-Relacional

- SQL e Java
  - SQL é a linguagem utilizada para interagir com o Banco de Dados Relacional. Em java utilizamos a API JDBC para execução de comandos SQL. Embora essa implementação seja eficaz, deixa a aplicação sensível a erros e dificulta as rotinas de testes
- Modelo OO e o Modelo Relacional
  - OO e Relacional são tecnologias conceitualmente diferentes. Um bom Design OO pode não ser um bom Modelo Relacional. Muitas vezes acabamos sacrificando o modelo OO por um melhor modelo Relacional
- Solução ideal
  - Ter os conceitos de OO representados no modelo relacional. Modelo OO independente da implementação Relacional. Aproveitando assim o melhor de cada tecnologia. MOR parece ser então a melhor solução para o problema !

Mapeamento OO-Relacional

- MOR é uma técnica que mapeia Objetos à tabelas no banco de dados, levando conceitos fundamentais de OO para modelo relacional
- Características de soluções MOR
  - Dirty Checking e Transitive Persistence
  - Lazy Fetching
  - Outer Join Fetching
  - Estratégias básicas de mapeamento de Herança
- Com MOR temos
  - Transparência na camada de persistência
  - Um melhor ambiente para execução de rotinas de testes
  - Desenvolvedor concentrado mais nas regras de negócio e menos no código relacionado a persistênca
  - Uma estratégia mais robusta quanto a mudanças nos modelos

Componentes de um Mecanismo de Persistência

- Linguagem de Consulta
  - Postergar gravações até o final das transações
    - **Inserir/atualizar/deletar menos registros**
    - **Utilizar apenas as colunas afetadas**
    - **Fim dos problemas de nível de isolamento de transações**
    - **Lock otimista realizado pelo mecanismo de persistência**
  - Problema: quando realizar leituras
    - **No acesso a uma propriedade do objeto?**
    - **Carga antecipada de objetos relacionados?**

Componentes de um Mecanismo de Persistência

- **O grande avanço dos bancos relacionais em relação às tecnologias anteriores (rede, hierárquica, ISAM) foi a linguagem de consulta declarativa**
  - Utilizar apenas o grafo de objetos é reverter para consultas realizadas de forma procedural
  - Mas um modelo OO não é um modelo Relacional – as consultas não são realizadas em termos de junções (joins), produtos cartesianos e projeções!

OQL x SQL

- **SQL necessita de muitos joins:**

```
select nome from produto p, venda v
where p.id = v.produto
```

- **OQL pode utilizar o grafo de objetos:**

```
select v.produto.nome from venda v
```

- **OQL pode utilizar operadores de conjunto:**

```
select v.cliente from vendas v, in v.produtos p
where sum(p.valor) > 10000
```

Componentes de um Mecanismo de Persistência

- **API de acesso**
  - Fornece os métodos para recuperação e atualização de objetos persistência
  - APIs intrusivas:
    - **Exigem que as classes estendam uma classe ou implementem uma interface**
  - APIs transparentes:
    - **Utilizam aspectos, manipulação de byte-codes ou pré-processamento para modificar dinamicamente as classes e inserir chamadas ao mecanismo de persistência**

Componentes de um Mecanismo de Persistência

- API de acesso
  - Basicamente, inserem um objeto persistente no contexto de uma transação e informam quando a transação é encerrada
  - Fornecem meios de recuperar objetos persistentes, mas também é possível utilzar os métodos getXXX dos próprios objetos
  - Em geral não há métodos explícitos para “gravar”, mas pode haver um factory para criar instâncias persistentes
  - Na prática, o desenvolvedor usa mais a API das suas classes do que do mecanismo de persistência

POO-Java

Padrões de Persistência de Objetos em Java

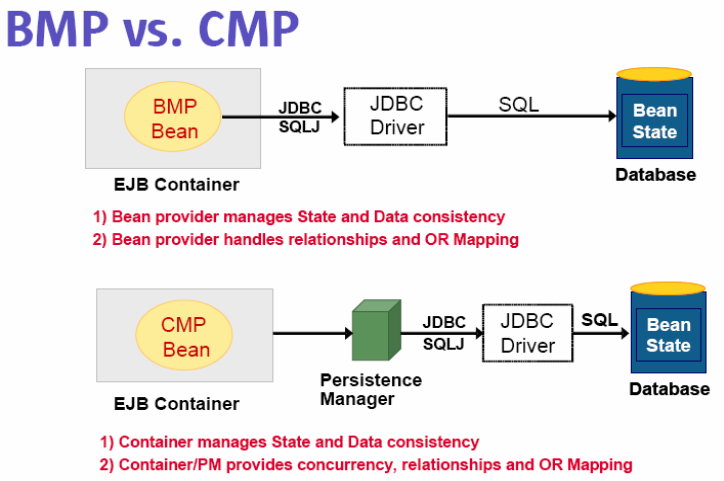
Padrões de Persistência em Java

- **Component-based approach**
  - **EJB EntityBeans**
    - **CMP (Container Managed Persistence)**
    - **BMP (Bean Managed Persistence)**
- **Object/relational approach**
  - **JDO – Java Database Object**
  - **Hibernate**

CMP e BMP

- API intrusiva, considera que um objeto persistente é antes disso um componente distribuído (remoto)
- Exige o uso de um servidor de aplicações
- Bastante maduro, com recursos avançados de otimização e gerenciamento na maioria dos produtos disponíveis no mercado
- Curva de aprendizado bastante longa
- O padrão atual (2.1) peca por não especificar como é o mapeamento OO-Relacional, gerando dependência em relação ao servidor de aplicação (descritores proprietários)

BMP x CMP



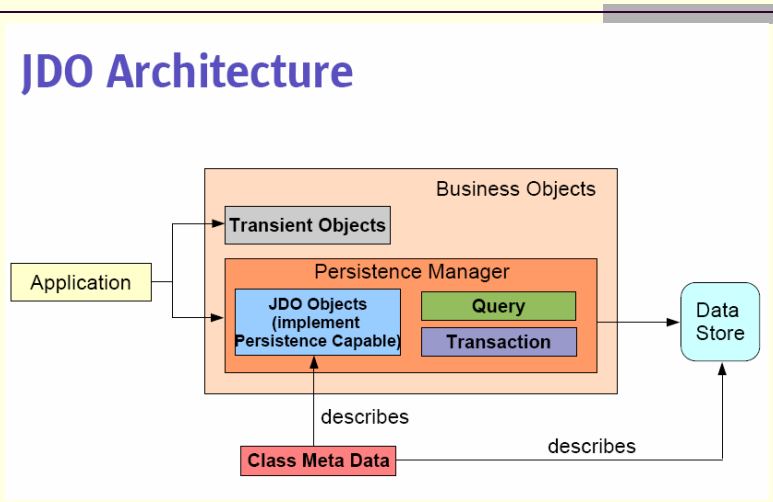
CMP e BMP

- Tem má fama no mercado por causa de limitações da versão 1.x, que não tinha recursos para relacionamentos entre Entity Beans
- A versão 3.0 é baseada no fato de que objetos persistentes não são expostos para a camada de apresentação (cliente) no modelo MVC
- Ficará semelhante ao Hibernate e ao JDO
- A nova versão também terá um padrão para o mapeamento OO/Relacional, compartilhado com o JDO 2.0

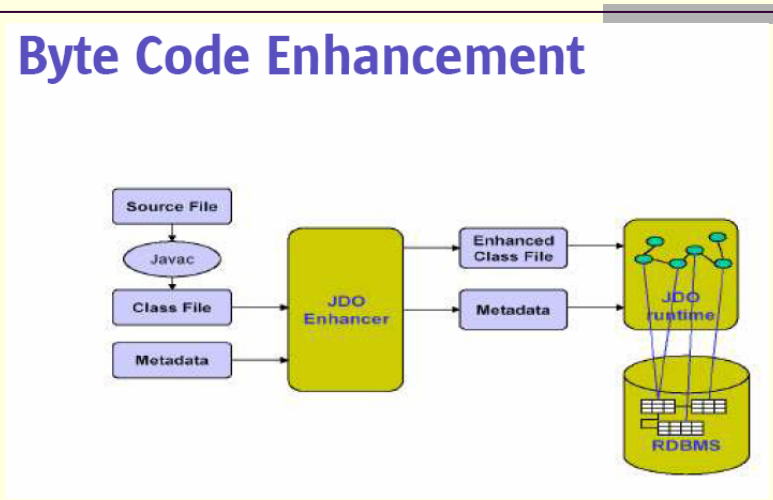
JDO

- Criado como alternativa ao CMP para aplicações que não rodam no servidor de aplicações
- Mas também é bem-integrado em alguns servidores de aplicações
- API não intrusiva, diferentes implementações utilizam manipulação de byte-codes ou pré-processamento
- O padrão atual peca por não definir o mapeamento OO/Relacional, mas o problema será resolvido na versão 2.0 da especificação

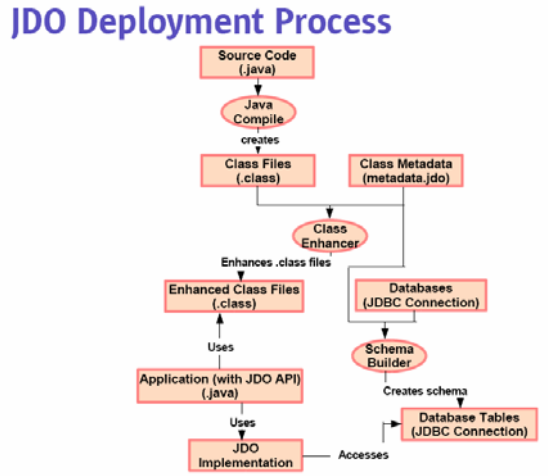
JDO Architecture



JDO Byte Code Enhancement



JDO Development Process



Hibernate

- Não é padrão do JCP, mas é quase um padrão de fato do mercado
- Ganhou muito espaço por causa do “preconceito” contra EJBs e da padronização incompleta do CMP e do JDO
- A versão 3.0 está melhorando muito a documentação e recursos de otimização
- Incorporado ao JBoss 4.0 como base do seu mecanismo CMP/CMR
- Famoso pela sua flexibilidade em termos de linguagem de consulta (HQL) e API

Hibernate APIs