

Módulo II

Persistência de Objetos

Prof. Ismael H F Santos

Ementa

- Arquitetura de camadas de Software
 - Arquiteturas em Camadas
 - Padrões para Arquiteturas em camadas
- Persistência de Objetos
 - Persistência de objetos
 - Estratégias de Persistência

FPSW-Java

Persistência
de Objetos



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

3

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 que correspondem àqueles que abstraem conceitos de negócio:
 - Num sistema de processamento de pedidos: Cliente, Pedido e Produto.
 - Numa aplicação financeira: Cliente, Conta, Crédito, Débito.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

4

Estratégias de Persistência

- Em aplicações OO, a persistência permite que objetos “sobrevivam” após o término da sessão em que foram criados.
 - Aplicações OO necessitam armazenar seus objetos persistentes em um mecanismo de armazenamento persistente.
 - Interpretação: o estado de um objeto pode ser armazenado em disco, para ser restaurado em outra sessão de uso.
- Há diversas estratégias que uma aplicação pode aplicar para alcançar a persistência de alguns de seus objetos.
- A estratégia adequada para uma certa aplicação depende da complexidade dessa aplicação.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

5

Estratégias de Persistência

- Vantagens do uso de uma camada de persistência separada:
 - Objetos do negócio podem ser reutilizados em aplicações que utilizam diferentes SGBD's.
 - Objetos de domínio se tornam mais fáceis de entender, já que não implementam acesso a banco de dados.
 - Melhores manutenibilidade e no reusabilidade.
- Há diversas estratégias (não exclusivas entre si) de persistência. Algumas delas são:
 - Serialização
 - SGBD Orientado a Objetos
 - Frameworks MOR
 - Padrões (Data Source Architectural Patterns): Active Record, Row Data Gateway, Table Data Gateway, DAO, Data Mapper

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

6

Uso de Serialização

- LP's modernas possuem funcionalidade para descarregar o estado da aplicação em um arquivo ou em um banco de dados. Todo o grafo de objetos de uma aplicação é *serializado* e armazenado, para futura restauração:
 - Prevailer (<http://www.prevailer.org/wiki.jsp>)
 - Bamboo.Prevalence-<http://bbooprevalence.sourceforge.net>
- Vantagem:
 - desenvolvedor trabalha em memória principal.
- Desvantagens:
 - Estado da aplicação deve ser gravado/restaurado todo de uma vez em memória principal.
 - Mudança do esquema do modelo OO, acesso concorrente...

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

7

Uso de um SGBD Orientado a Objetos

- Por que não Bancos de Dados OO?
 - Um SGBD orientado a objetos possui o conceito de classe definido internamente. Alguns exemplos:
 - Versant (<http://www.versant.com/>)
 - Ozone (<http://www.ozone-db.org/>)
 - db4objects, Inc. (<http://www.db4objects.com/>)
 - Entretanto 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. Apesar de padronizado pela OMG: <http://www.odmq.org>, a tecnologia não se desenvolveu comercialmente.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

8

Uso de um SGBD Relacionais

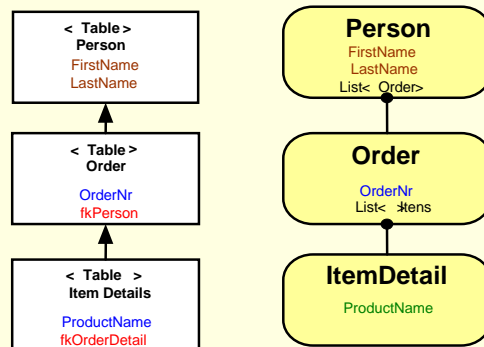
- **Porque Bancos de Dados Relacionais ?**
 - Têm se mostrado bastante eficientes, o que levou a uma boa aceitação do mercado.
 - Algumas aplicações utilizam-se de Stored Procedures e Triggers para execução de regras de negócio.
 - Atualmente grande parte da Aplicações utilizam e ou integram-se por meio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

Estratégias de MOR

- **Relevância atual do mapeamento de objetos para o modelo relacional:**
 - A OO é a forma usual de desenvolver sistemas de software.
 - Por outro lado, os SGBD relacionais dominam o mercado.
- **Os princípios básicos são bastante diferentes nos dois “mundos”.**
 - **MO:** elementos (objetos) são abstrações de comportamento e informações.
 - **MR:** elementos representam informações no formato tabular.
- **A universalidade (padronização) da SQL não ajuda tanto na solução do problema.**

Descasamento de informações

- O descasamento de informações (*impedance mismatch*) corresponde ao problema de mapear as representações do modelo de objetos e do modelo relacional.



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

11

FPSW-Java

Mecanismos de
Persistência
de Objetos



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

12

Componentes de um Mecanismo de Persistência

- **Mapeamento OO/Relacional (MOR) ou *Object-relational mapping***
- **Linguagem de Consulta**
- **API de acesso aos dados**

Componentes de um Mecanismo de Persistência

- **Mapeamento OO/Relacional (MOR)**
 - O mapeamento objeto relacional é o procedimento de automatizar a mapeamento dos objetos em tabelas de um SGBD relacional.
 - Um modelo OO é semanticamente **mais rico** do que um modelo Relacional. Vários modelos Relacionais são possíveis para um mesmo modelo OO
 - 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.
 - Objetivos de performance e confiabilidade podem levar a mapeamentos “degenerados”, semelhantes a bancos relacionais não normalizados.

Componentes de um Mecanismo de Persistência

■ Mapeamento OO/Relacional (MOR)

- 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 a codificação de rotinas de testes.
- 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 !

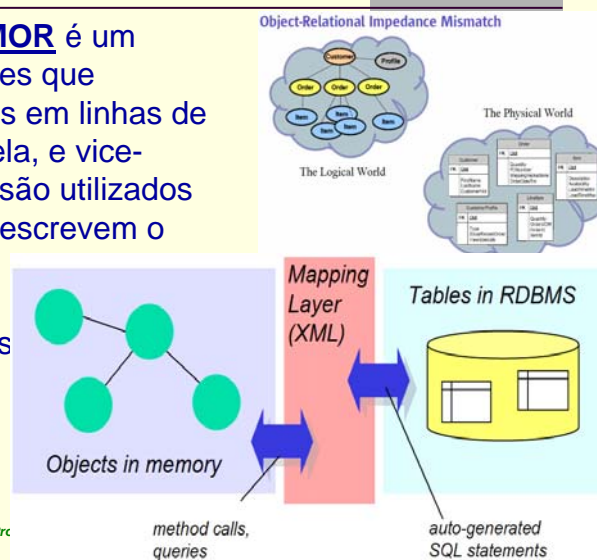
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

15

Frameworks para MOR

- Um **framework MOR** é um conjunto de classes que transforma objetos em linhas de uma ou mais tabela, e vice-versa. Para isso, são utilizados metadados que descrevem o mapeamento entre classes do modelo de objetos e tabelas do banco de dados.



April 05

Prc

Recursos de um Framework MOR

- Framework MOR
 - **Benefícios:** Produtividade, Manutenibilidade, Independência de Vendedor (de SGBD), Minimiza LOC (Lines of Code)
 - **Desvantagem:** curva de aprendizado, desempenho (?).
- Outros recursos:
 - Uma API para realizar operações **CRUD** sobre objetos.
 - Uma linguagem de consulta sobre o esquema de objetos (em vez do esquema do banco de dados).
 - Facilidade para permitir a definição do mapeamento de objetos para relações no banco de dados via arquivos de configuração.
 - manipulação de objeto transacionais
 - Associação tardia (Lazy Fetching e Lazy association),
 - Gerência de cache

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

17

Mapeamento OO-Relacional

- manipulação de objeto transacionais (cont.)
 - Gerência de *cache*.
 - Dirty Checking e Transitive Persistence
 - Outer Join Fetching
 - Estratégias básicas de mapeamento de Herança
- Dessa forma com MOR temos
 - Transparência na camada de persistência
 - Um melhor ambiente para execução de testes
 - Desenvolvedor concentrado mais nas regras de negócio e menos no código relacionado a persistência
 - Estratégia mais robusta quanto a mudanças nos modelos

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

18

Exemplos de Frameworks para MOR

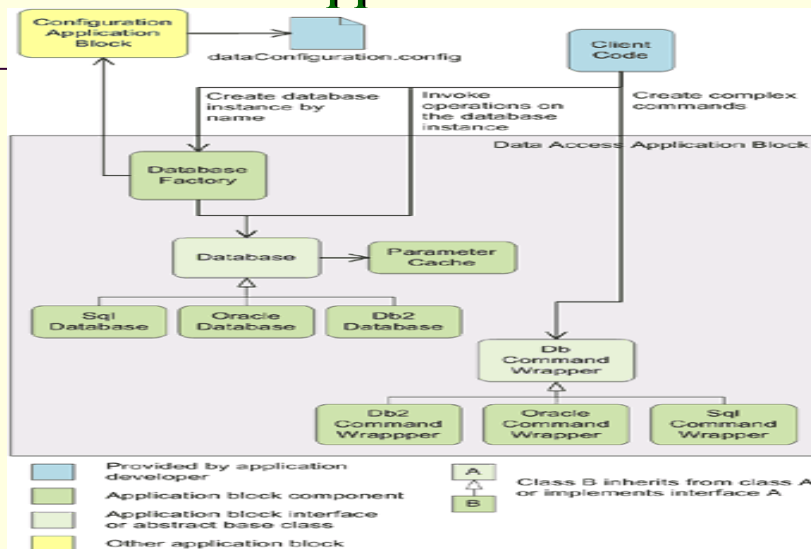
- Alguns exemplos de frameworks para MOR:
 - JPA – padronização para frameworks MOR
 - Hibernate
 - NHibernate
 - JDO
 - ObjectRelationalBridge
 - Castor JDO
 - iBatis SQL Maps
 - Toplink (comercial)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

19

Data Access Application Block



<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/daab.asp>

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

20

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

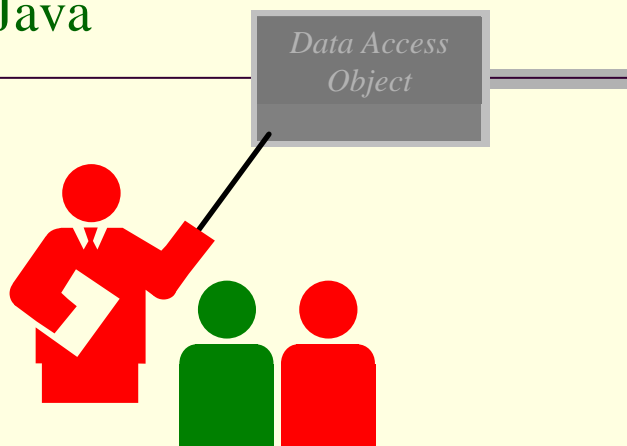
- **API de acesso**
 - Fornece os métodos para recuperação e atualização de objetos persistência
 - APIs intrusivas:
 - **Exigem que as classes estendam uma classe ou implementem uma interface**
 - APIs transparentes:
 - **Utilizam aspectos, manipulação de byte-codes ou pré-processamento para modificar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

FPSW-Java



Patterns

- Não existe uma definição exata para o que são patterns. O autor **Martin Fowler**, define patterns como uma idéia que foi útil em um contexto prático e que provavelmente será útil em outros. Outros autores definem como uma regra que expressa uma relação entre um contexto, um problema e uma solução. Mas em geral, patterns tem sempre as seguintes características:
 - são notados através da experiência;
 - evitam que se reinvente a roda;
 - existem em diferentes níveis de abstração;
 - são artefatos reutilizáveis;
 - passam aos desenvolvedores designs corretos e
 - podem ser combinados para resolver um grande problema;
 - aceitam melhoramentos contínuos.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

27

Pattern Data Access Object (DAO)

- **Objetivo**
 - **Abstrair e encapsular todo o acesso a uma fonte de dados. O DAO gerencia a conexão com a fonte de dados para obter e armazenar os dados.**
- O padrão **Data Access Object**, também conhecido como o padrão **DAO**, abstrai a recuperação dos dados tal com com uma base de dados. O conceito é "separar a relação do cliente de um recurso dos dados de seu mecanismo de acesso dos dados."
- O **DAO** é utilizado para encapsular a lógica de acesso a dados. Assim, se for necessário a alteração de banco de dados, não é necessário alterar todo sistema, mas somente os DAOs.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

28

Pattern DAO

- Dentro do **DAO** são realizadas as queries ou o acesso aos métodos do Hibernate. A intenção real de existência dos **DAOs** é que eles não possuam nenhuma lógica de negócio, apesar de algumas vezes ser necessário encapsular algo dentro deles, especialmente quando outros patterns da camada de modelo não estão presentes.
- Quando utilizado junto com **Hibernate**, ambos realizam o trabalho de abstrair a base, pois o Hibernate já mascara o tipo do banco de dados, ficando para o **DAO** a parte de controlar as conexões, exceções, retornos para os níveis superiores, entre outros.

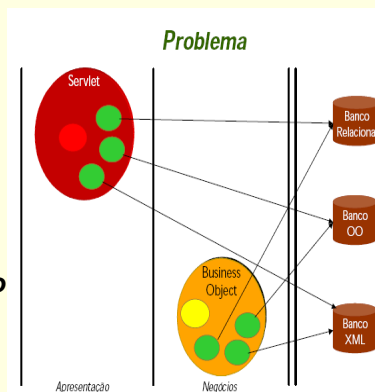
April 05

Prof. Ismael H. F. Santos - ismael@tegraf.puc-rio.br

29

Problema

- **Forma de acesso aos dados varia consideravelmente dependendo da fonte de dados usado**
 - Banco de dados relacional
 - Arquivos (XML, CSV, texto, formatos proprietários)
 - LDAP
- **Persistência de objetos depende de integração com fonte de dados (ex: business objects)**
 - Colocar código de persistência (ex: JDBC) diretamente no código do objeto que o utiliza ou do cliente amarra o código desnecessariamente à forma de implementação
 - Ex: difícil passar a persistir objetos em XML, LDAP, etc.



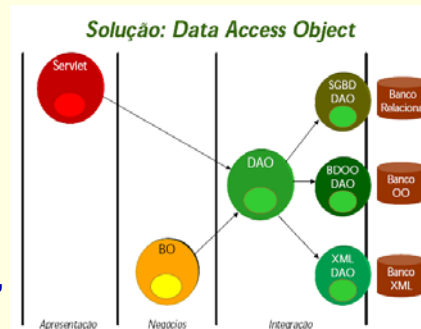
April 05

Prof. Ismael H. F. Santos - ismael@tegraf.puc-rio.br

30

Solução

- **Data Access Object (DAO)** oferece uma interface comum de acesso a dados e esconde as características de uma implementação específica
 - Uma API: métodos genéricos para ler e gravar informação
 - Métodos genéricos para concentrar operações mais comuns (simplificar a interface de acesso)
- DAO define uma interface que pode ser implementada para cada nova fonte de dados usada, viabilizando a substituição de uma implementação por outra
- DAOs não mantêm estado nem cache de dados



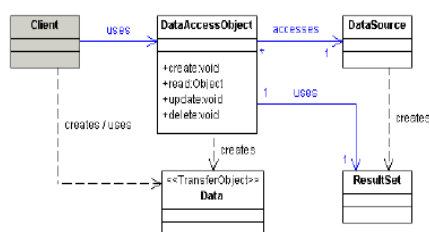
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

31

UML

- **Client:** objeto que requer acesso a dados: **Business Object, Session Façade, Application Service, Value List Handler, ...**
- **DAO:** esconde detalhes da fonte de dados
- **DataSource:** implementação da fonte de dados
- **Data:** objeto de transferência usado para retornar dados ao cliente. Poderia também ser usado para receber dados.
- **ResultSet:** resultados de pesquisa no banco



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

32

Estratégias de Implementação

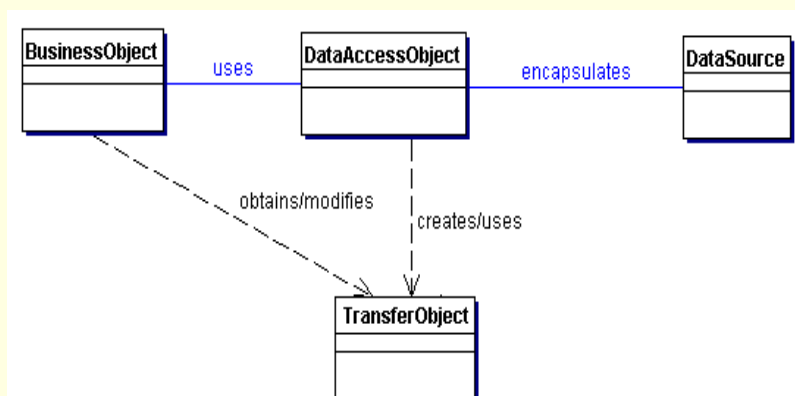
- **Custom DAO Strategy**
 - *Estratégia básica. Oferece métodos para criar, apagar, atualizar e pesquisar dados em um banco.*
 - *Pode usar **Transfer Object** para trocar dados com clientes*
- **DAO Factory Method Strategy**
 - *Utiliza **Factory Methods** em uma classe para recuperar todos os DAOs da aplicação*
- **DAO Abstract Factory Strategy**
 - *Permite criar diversas implementações de fábricas diferentes que criam DAOs para diferentes fontes de dados*

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

33

DAO – Padrao Catalogo do JEE

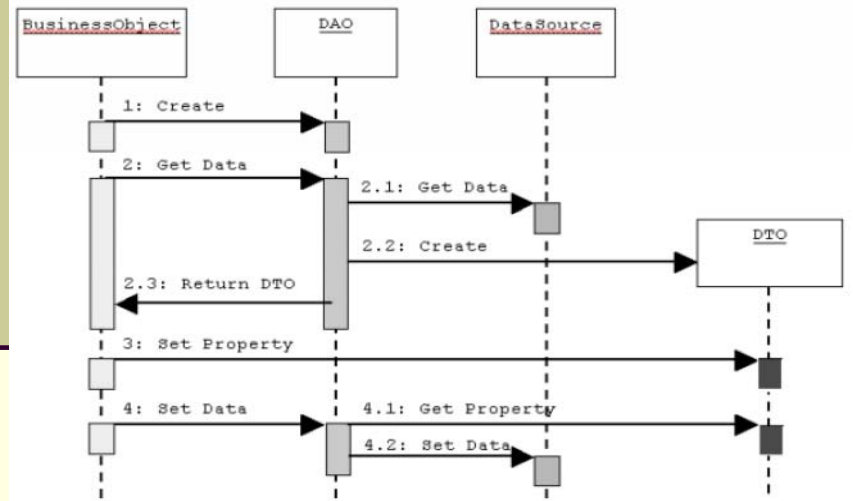


April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

34

DAO - interação



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

35

Interface CustomerDAO

```
public interface CustomerDAO {
    public int insertCustomer(...);
    public boolean deleteCustomer(...);
    public Customer findCustomer(...);
    public boolean updateCustomer(...);
    public RowSet selectCustomersRS(...);
    public Collection selectCustomersTO(...);
    ...
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

36

CloudscapeCustomerDAO impl

```
/* This class can contain all Cloudscape specific code and SQL
statements. The client is thus shielded from knowing these
implementation details */
import java.sql.*;
public class CloudscapeCustomerDAO implements CustomerDAO {
    public CloudscapeCustomerDAO() { // initialization }
    /* The following methods can use CloudscapeDAOFactory.
    createConnection() to get a connection as required */
    public int insertCustomer(...) {
        /* Implement insert customer here. Return newly created customer
        number or a -1 on error */
    }
    public boolean deleteCustomer(...) {
        /* Implement delete customer here // Return true on success,
        false on failure */
    }
}
April 05 Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br 37
```

CloudscapeCustomerDAO impl

```
public Customer findCustomer(...) {
    /* Implement find a customer here using supplied argument values as
    search criteria. Return a Transfer Object if found, return null on error or
    if not found */
}
public boolean updateCustomer(...) {
    /* implement update record here using data from the customerData
    Transfer Object Return true on success, false on failure or error */
}
public RowSet selectCustomersRS(...) {
    /* implement search customers here using the supplied criteria.
    Return a RowSet. */
}
public Collection selectCustomersTO(...) {
    /* implement search customers here using the supplied criteria.
    Alternatively,
    implement to return a Collection of Transfer Objects. */
}
}
April 05 Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br 38
```

Client Code

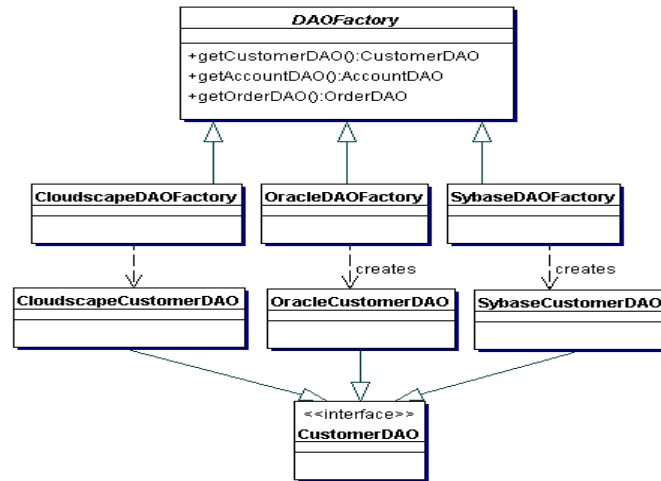
```
// Create the required DAO Factory and the DAO Object
DAOFactory cloudscapeFactory =
    DAOFactory.getDAOFactory(DAOFactory.DAO_CLOUDSCAPE);
CustomerDAO custDAO = cloudscapeFactory.getCustomerDAO();
// Create a new customer. Find a customer object. Get Transfer Object.
int newCustNo = custDAO.insertCustomer(...);
Customer cust = custDAO.findCustomer(...);
// modify the values in the Transfer Object. Update the customer object
cust.setAddress(...); cust.setEmail(...); custDAO.updateCustomer(cust);
// delete a customer object
custDAO.deleteCustomer(...);
// select all customers in the same city
Customer criteria=new Customer(); criteria.setCity("New York");
Collection customersList = custDAO.selectCustomersTO(criteria);
// returns collection of CustomerTOs. iterate through this collection to get
values. ...
April 05 Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br 39
```

CustomerTransferObject

```
import java.io.*;
public class Customer implements Serializable {
    // member variables
    int CustomerNumber;
    String name;
    String streetAddress;
    String city;
    ...
    // getter and setter methods...
    ...
}
```

April 05 Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br 40

Using Abstract Factory Pattern

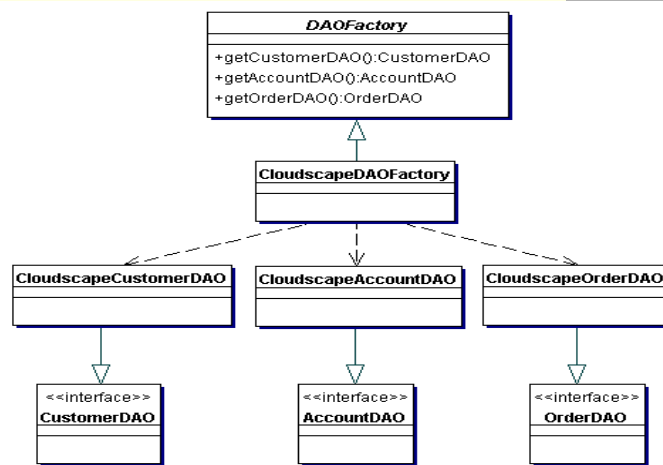


April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

41

Implementing Factory for DAOStrategy Using Factory Method Pattern



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

42

Abstract class DAO Factory

```
public abstract class DAOFactory {
    // List of DAO types supported by the factory
    public static final int CLOUDSCAPE = 1;
    public static final int ORACLE = 2;
    public static final int SYBASE = 3; ...
    /* There will be a method for each DAO that can be created. The concrete
       factories will have to implement these methods. */
    public abstract CustomerDAO getCustomerDAO();
    public abstract AccountDAO getAccountDAO();
    public abstract OrderDAO getOrderDAO(); ...
    public static DAOFactory getDAOFactory( int whichFactory) {
        switch (whichFactory) {
            case CLOUDSCAPE: return new CloudscapeDAOFactory();
            case ORACLE : return new OracleDAOFactory();
            case SYBASE : return new SybaseDAOFactory(); ...
            default : return null;
        }
    }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

43

Cloudscape concrete DAO Factory implementation

```
public class CloudscapeDAOFactory extends DAOFactory {
    public static final String DRIVER= "COM.cloudscape.core.RmiJdbcDriver";
    public static final String
    DBURL="jdbc:cloudscape:rmi://localhost:1099/CoreJ2EEDB";
    // method to create Cloudscape connections
    public static Connection createConnection() { .. // create a connection }
    public CustomerDAO getCustomerDAO() {
        // CloudscapeCustomerDAO implements CustomerDAO
        return new CloudscapeCustomerDAO();
    }
    public AccountDAO getAccountDAO() {
        // CloudscapeAccountDAO implements AccountDAO
        return new CloudscapeAccountDAO();
    }
    public OrderDAO getOrderDAO() { // implements OrderDAO ...
        return new CloudscapeOrderDAO();
    }
}

***April 05
```

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

44

Conseqüências DAO

- Transparência quanto à fonte de dados
- Facilita migração para outras implementações
 - Basta implementar um DAO com mesma interface
- Centraliza todo acesso aos dados em camada separada
 - Qualquer componente pode usar os dados (servlets, componentes de negocio, etc)
- Camada adicional
 - Pode ter pequeno impacto na performance
- Requer design de hierarquia de classes (Factory)
- Exemplos de DAO
 - DAO para cada Business Object
 - DAO para serviços arbitrários

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

45

POO-Java

*Padrões de
Persistência de
Objetos em Java*



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

46

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

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

47

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 2.1 pecava por não especificar como é o mapeamento OO-Relacional, gerando dependência em relação ao servidor de aplicação (descritores proprietários)

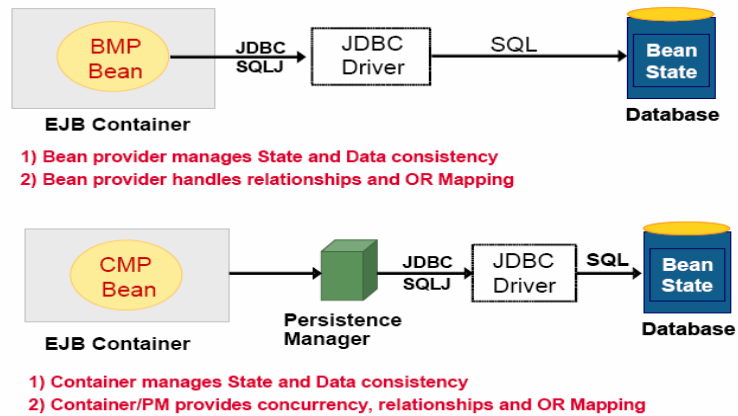
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

48

BMP x CMP

BMP vs. CMP



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

49

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

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

50

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

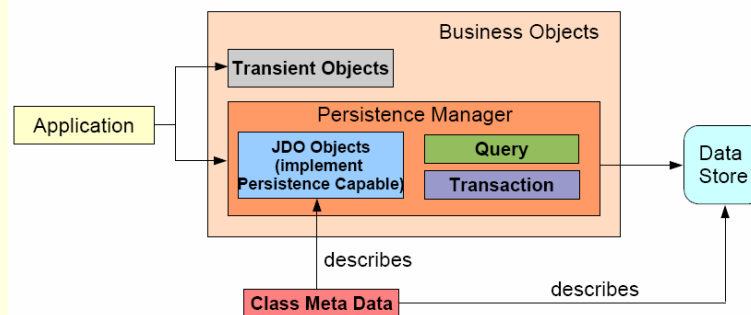
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

51

JDO Architecture

JDO Architecture



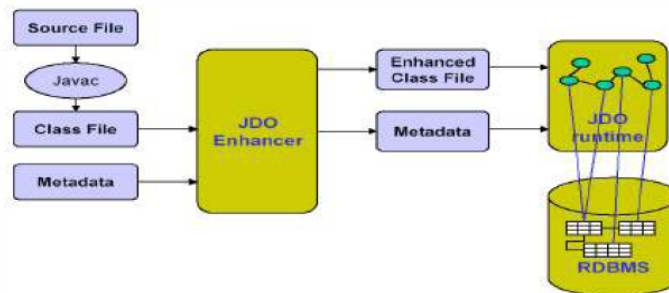
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

52

JDO Byte Code Enhancement

Byte Code Enhancement



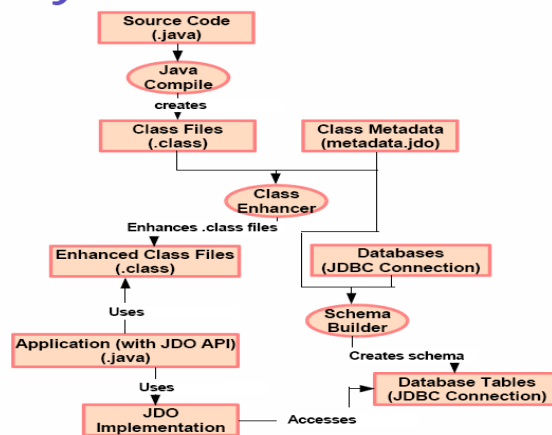
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

53

JDO Development Process

JDO Deployment Process



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

54

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

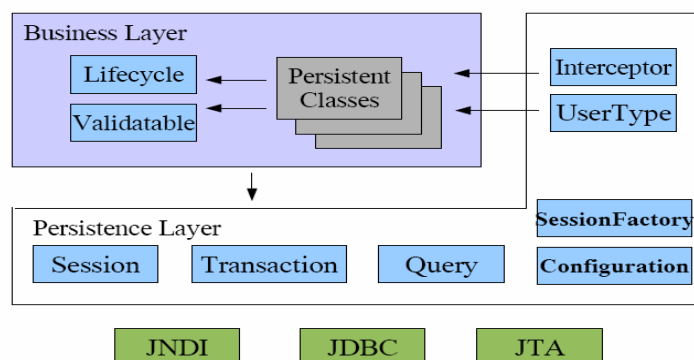
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

55

Hibernate APIs

Hibernate APIs



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

56