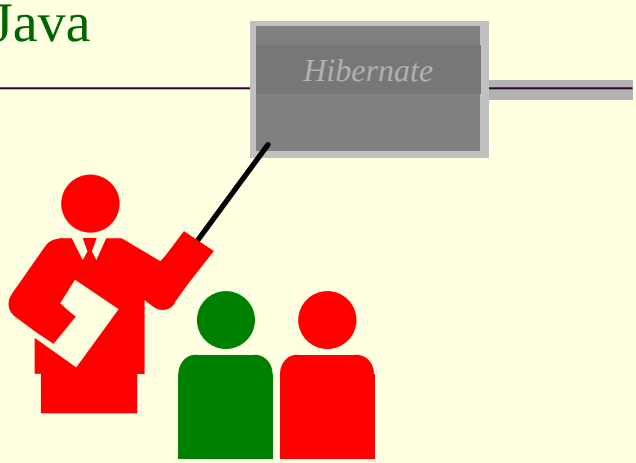
Módulo II
Persistência com Hibernate

Prof. Ismael H F Santos

Ementa

- Persistência com Hibernate
  - Hibernate
  - Mapeamento OO-Relacional no Hibernate
  - Configuração do Hibernate
  - Mapeamento de Classes
  - Hibernate Query Language
  - Exemplo de Aplicação

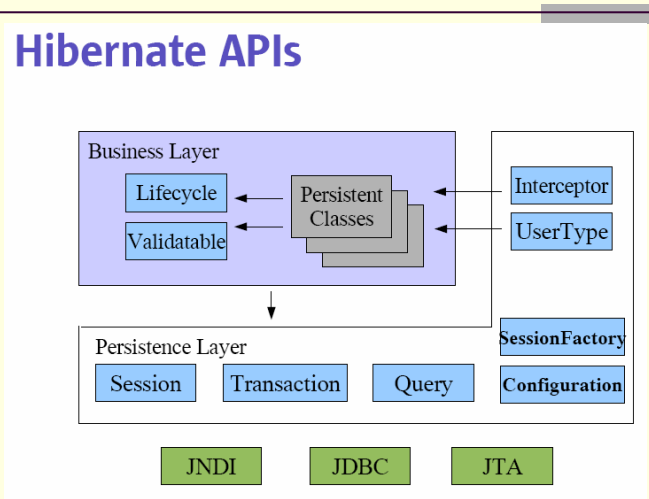
FPSW-Java



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



Hibernate

- Framework de persistência MOR.
- Um dos mais bem sucedidos e documentados projetos open-source.
- Vantagens
  - Modelo natural de programação OO, incluindo herança, composição, polimorfismo e relacionamentos com a Collection API
  - Transparência de Bando de Dados
  - Modelo OO independente da implementação relacional
  - Performace
  - Dois níveis de cache com suporte a diferentes implementações
  - Simplicidade com POJOs (Plain Old Java Objects)
  - Integração com JTA
  - Comunidade
  - Documentação
  - Ferramentas

Classe Persistente para o Hibernate

- Um simples POJO mapeado a tabelas no banco através de um arquivo XML.
- **POJO (Plain Old Java Object)**
  - Classe que representa a entidade
  - Propriedades, métodos de negócio e relacionamentos
  - Exemplo: A entidade Projeto
- Arquivo de mapeamento
  - Algoritmos para geração de chave primária
  - Nome das tabelas e colunas
  - Constraints e Índices
  - Relacionamentos e estratégias de extração de dado
  - Política de cascade
  - Formulas
  - Configurações de comportamento de persistência

Hibernate.properties File

hibernate.properties

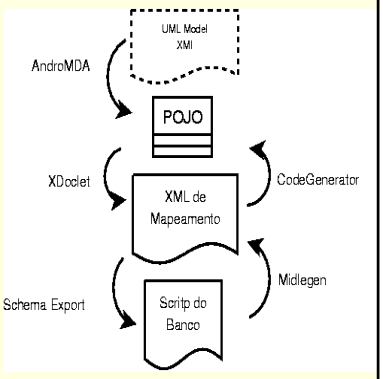
```
<class
   name = "User"
   table = "USER">
   <property
      name = "username"
      column = "USERNAME"
      type = "string"/>
   <component
      name = "homeAddress"
      class = "Address">
      property name = "street"
               type = "string"
               column = "HOME_STREET"
               notnull = "true"/>
      ...
   </component>
   ...
</class>
```

Entity Type – has its own database identity

Value Type – has no database identity and persistent state dependent on the owning identity

Artefatos do Hibernate

- Para o Hibernate existem 3 artefatos principais, o POJO, o XML de Mapeamento e o Schema do banco de dados.
- Ferramentas
  - A idéia é que com qualquer um destes artefatos, seja possível construir os outros dois utilizando ferramentas.
- AndroMDA - Model Driven Achitecture
  - Independente da implementação
  - Dependente da implementação



Hibernate

- As principais interfaces do Hibernate são
  - Session
  - Transaction
  - SessionFactory
  - Configuration
  - Query e Criteria API

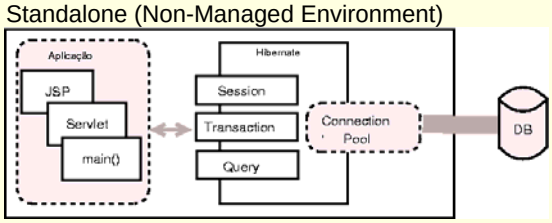
Hibernate

- Analisando a entidade Projeto
  - O único requerimento do Hibernate é a implementação do construtor default
  - Apesar de ser possível mapear um atributo público, é uma boa prática deixar o atributo privado e criar os métodos de acesso setXXX() e getXXX()
- O XML de mapeamento
  - Além das informações sobre o mapeamento do objeto, podemos configurar comportamentos relacionados a persistência da entidade, atributos e relacionamentos

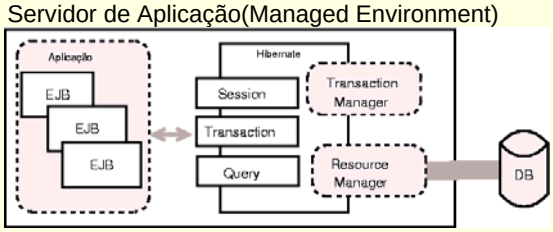
Arquitetura do Hibernate

- 2 cenários de utilização do Hibernate:
  - standalone
  - integrado ao servidor de aplicação

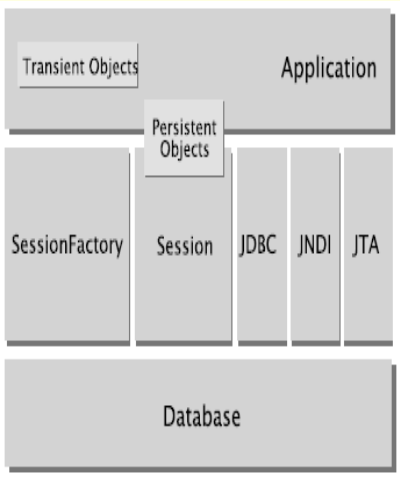
Standalone (Non-Managed Environment)



Servidor de Aplicação(Managed Environment)

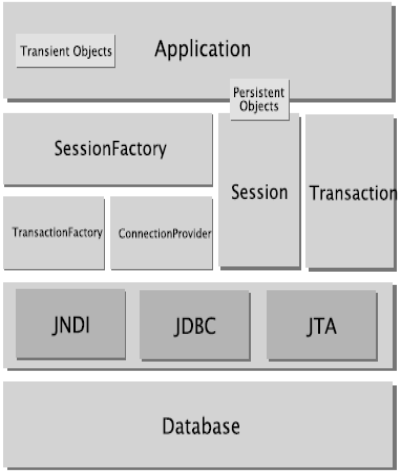


Arquitetura do Hibernate



- Modelo 1
  - Aplicação fornece as conexões JDBC;
  - Aplicação gerencia as transações;

Arquitetura do Hibernate



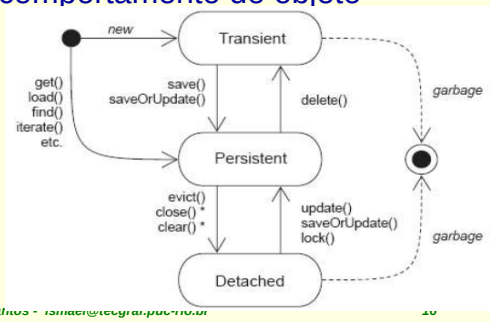
- Modelo 2
  - Aplicação não gerencia as transações e não fornece as conexões JDBC;
  - Hibernate se integra com as API's de gerenciamento de transações e conexões JDBC do servidor de aplicações;

Arquitetura do Hibernate

- Persistent Objects / Collections
  - Objetos que possuem estado persistente;
  - Podem ser simples JavaBeans;
  - Estão associados a um objeto Session;
- Transaction (net.sf.hibernate.Transaction)
  - Aplicação é “desacoplada” da estratégia de transação (JDBC/JTA/CORBA) a ser utilizada;
  - Um Session pode abrir várias transações;
- ConnectionProvider (net.sf.hibernate.connection.ConnectionProvider)
  - Uma fábrica (e pool) de conexões JDBC;
  - Aplicação é “desacoplada” do Datasource ou DriverManager utilizado;
- TransactionFactory (net.sf.hibernate.TransactionFactory)
  - Uma fábrica de instâncias da classe Transaction;

Estados de um objeto persistente

- Uma instância de um objeto persistente pode estar em um deste três estados:
  - Transient, Persistent ou Detached
- Para um bom desenvolvimento com Hibernate é fundamental entender o comportamento do objeto em cada estado, assim como os eventos que causam a transição de um estado para outro.



Hibernate API

- Core Interfaces API
  - Em Core Interfaces encontramos os principais componentes do Hibernate que são: Session, SessionFactory, Transaction, Configuration, Query e Criteria.
- Callback Interfaces
  - Temos três interfaces nesta categoria, Validatable, Lifecycle e Interceptor.
  - Validatable e Lifecycle permite que a entidade receba informações relacionadas a sua própria persistência
  - Com Interceptor podemos, por exemplo, realizar auditoria sobre as operações realizadas com determinada classe persistente.

Hibernate API

- Types
  - Hibernate suporta todos os tipos primitivos assim como as principais classes do java, como Calendar, Date, String, etc
  - Utilizando a interface UserType podemos criar um novo tipo de dado. Como por exemplo um tipo de dado que realize sua persistência em duas colunas na tabela.
- Pontos de extensão
  - Com esta API é possível extender as funcionalidades e estratégias do Hibernate

Hibernate API

- Definições:
  - SessionFactory (net.sf.hibernate.SessionFactory)
    - Threadsafe;
    - Cache de mapeamentos objeto-relacional;
    - Cliente do Connection-Provider;
  - Session (net.sf.hibernate.Session)
    - Representa a interação entre a aplicação e o meio de persistência;
    - Encapsula uma conexão JDBC;
    - Uma fábrica (factory) de transações;

Transação e Concorrência

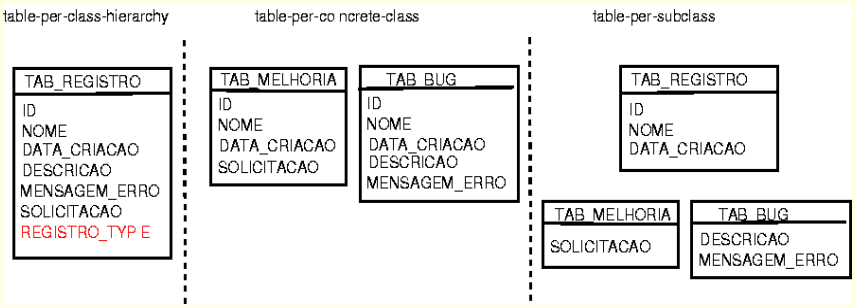
- Transação
  - Com JDBC API podemos iniciar uma transação através do método setAutoCommit(false) da interface Connection
  - Em alguns casos temos que trabalhar com transações distribuídas, por exemplo, se utilizarmos dois banco de dados. Neste caso temos que utilizar um Transaction Manager que controla a distribuição e o commit e rollback das transações.
  - Hibernate oferece uma camada de transparência relacionada à transação
  - Uma chamada a session.beginTransaction() pode resultar em uma JDBC Transaction ou JTA Transaction
- Lock
  - É um mecanismo que permite controlar o acesso concorrente a um registro
  - Hibernate permite pessimistic locking (SELECT FOR UPDATE) ou optimistic locking

FPSW-Java

Mapeamento OO-Relacional no Hibernate

Mapeamento de Herança

- Oferece 3 estratégias de mapeamento de herança
  - Table-per-class-hierarchy
  - Table-per-concrete-class
  - Table-per-subclass



Dirty Checking Transitive Persistence

- Dirty Checking
  - Quando uma entidade é carregada do banco de dados e alterada, Hibernate executa update apenas das colunas que realmente foram alteradas

```
Projeto projeto = (Projeto) session.load(Projeto.class,
                                         new Long(1)) ;
projeto.setNome("Alterado")
```

--> update projeto set nome = 'Alterado'

Dirty Checking Transitive Persistence

- Transitive Persistence
  - Com Transitive Persistence, se uma Collection que representa um relacionamento é alterada, ou seja, foi adicionado e/ou removido um Objeto, Hibernate automaticamente insere e/ou remove os registros na tabela do banco de dados, de acordo com sua política de cascade

projeto.addRegistro(new Bug("Novo Bug") ;

--> insert into bug ...
--> update bug set projeto = 1

Extração de dados com Hibernate

- Hibernate nos oferece três opções para extração de dados:
  - Hibernate Query Language (uma linguagem muito parecida com SQL), a Criteria API ou ainda com SQL nativo.
- Hibernate Query Language
  - Linguagem utilizada para extração de dados Orientada a Objetos
  - HSQL oferece quase tudo o que você encontra em SQL de like, upper(), sum(), avg(), in, some, group by até funções especiais para trabalhar com Collection.
  - Sub queries em clausulas where
  - Outer, left e implícito join

Extração de dados com Hibernate

- Hibernate Query Language (cont.)
  - Named Parameter e Parameter List deixam a query mais simples e legível
  - Com Projection é possível extrair individualmente as propriedades da entidade
  - Dynamic Instantiation deixa seu relatório mais simples e organizado
- Query Polimôrficas
  - Com queries polimórficas podemos extrair a entidades de tabelas diferentes fazendo uma única query

Extração de dados com Hibernate

- Criteria API
  - Uma mais orientada a objetos de extração de dados
  - Onde por exemplo um like se transforma em uma chamada ao método Expression.like() e um join em um createAlias()
- Query Nativa
  - Hibernate oferece suporte a execução de query nativas
  - Com este suporte, não há a necessidade de amarrar a query com nome de tabelas e colunas

Extração de dados com Hibernate

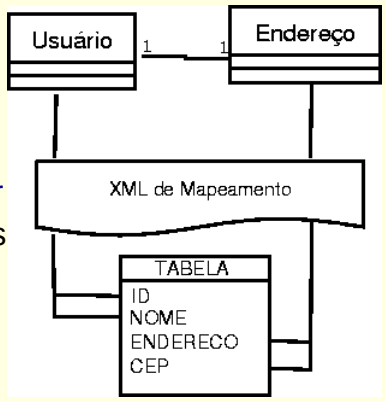
- Paginação com setFirstResult() e setMaxResult()
  - Para todas as opções de consulta com Hibernate, temos disponível a paginação
  - A implementação da paginação depende do banco de dados que se esta utilizando, mas para a aplicação é transparente.
  - No caso do Oracle por exemplo, Hibernate controla a paginação utilizando rownum

Relacionamentos entre entidades

- Tipos de relacionamentos
  - Unidirecional e Bidirecional
- Cardinalidade
  - Um para muitos ou muitos para um
  - Relacionamento muitos para muitos
- Relacionamento Polimórfico
- CMP 2.0 e Hibernate
  - Diferentemente de CMP 2.0, Hibernate não implementa CMR (Container Management Relationship)
  - É uma boa prática a implementação de métodos conveniente para relacionar entidades

Granularidade

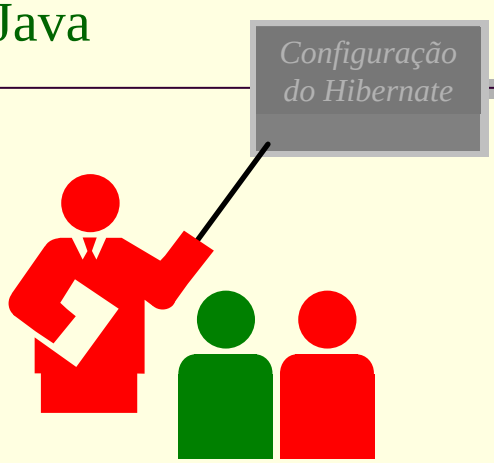
- Com Hibernate temos o design OO independente da implementação relacional, dessa forma podemos ter granularidade em OO mapeada a uma única tabela no BD.
- Diferença entre Entidade e Valor
  - Entidade é independente, pos um único ID e pode ser trabalhada diretamente
  - Valor tem seu ciclo de vida dependente da Entidade relacionada.



Extração de dados e relacionamentos

- Para a uma entidade que possui relacionamento, podemos escolher entre 4 estratégias Immediate, Eager, Lazy e Batch fetching, que define como as entidades relacionadas serão carregadas.
  - Immediate Fetching
    - Carrega as entidades relacionadas imediatamente, utilizando um select na seqüência do outro.
  - Eager Fetching
    - Indica para o Hibernate que as entidades devem ser carregadas utilizando um único acesso ao banco, o que implica na utilização de um outer join.
  - Lazy Fetching
    - Carrega somente a entidade principal, e quando necessário, as entidades relacionadas.
  - Batch Fetching
    - É uma técnica que permite melhoria na estratégia Lazy.

FPSW-Java



Configuração do Hibernate

- Uma aplicação pode ser configurada para utilizar diversos bancos de dados;
- O objeto Configuration é responsável pelo “parsing” dos mapeamentos objeto-relacionais declarados nos arquivos *.xml;
- O objeto Configuration pode ser instanciado diretamente pela aplicação;

Configuração do Hibernate

- Um objeto SessionFactory é construído a partir do objeto Configuration; A partir de sua construção não é mais necessária a utilização do objeto Configuration;
- A Configuração do SessionFactory pode ser feita das seguintes maneiras:
  - Através do arquivo hibernate.properties (raiz do classpath);
  - Através do arquivo hibernate.cfg.xml (raiz do classpath);
  - Via programação;

Configuração do Hibernate

- Cada SessionFactory deve ser configurado a partir de um único arquivo de configuração xml;
- O SessionFactory pode abrir novas sessões a partir de uma conexão JDBC fornecida pelo usuário;
- As conexões JDBC podem ser obtidas pelo próprio Hibernate, a configuração das conexões deverá ser feita através dos arquivos de configuração (hibernate.properties ou hibernate.cfg.xml);

Configuração do Hibernate

- Principais parâmetros para a configuração das conexões JDBC:
  - hibernate.connection.driver_class
  - hibernate.connection.url
  - hibernate.connection.username
  - hibernate.connection.password
  - hibernate.connection.pool_size
- O Hibernate pode utilizar as seguintes implementações de connection pooling: C3P0, Apache DBCP e Proxool;

Configuração do Hibernate

- Em servidores de aplicação o Hibernate pode obter conexões através de datasources registrados via JNDI.
- Os principais parâmetros são:
  - hibernate.connection.datasource -> Nome JNDI do datasource;
  - hibernate.jndi.url -> URL do JNDI provider;
  - hibernate.jndi.class (opcional) -> Classe factory do InitialContext
  - hibernate.connection.username
  - hibernate.connection.password

Configuração do Hibernate

- Outros parâmetros de configuração do Hibernate:
  - hibernate.dialect
  - hibernate.default_schema -> coloca automaticamente o nome do esquema antes do nome dos objetos do bd durante a geração do SQL;
  - hibernate.session_factory_name -> Nome JNDI que o session factory será registrado;
  - hibernate.use_outer_join -> Utiliza outer join sempre que possível;

Configuração do Hibernate

- hibernate.connection.provider_class -> Nome da classe ConnectionProvider;
- hibernate.cache.provider_class -> Nome da classe CacheProvider;
- hibernate.transaction.factory_class -> Nome da classe TransactionFactory;
- jta.UserTransaction -> Nome JNDI utilizado pela classe JTATransactionFactory para obter a classe javax.transaction.UserTransaction
- hibernate.show_sql -> Exibe os SQL's gerados pelo Hibernate;

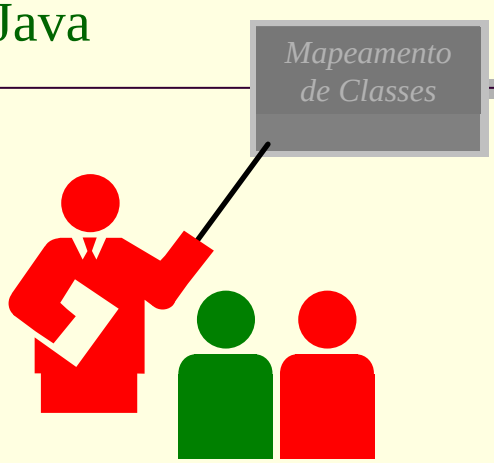
Configuração do Hibernate

- SQL Dialects
  - Dialetos SQL possibilitam que o Hibernate tire proveito de características próprias dos SGBD's;
  - Principalmente no caso da utilização de mecanismos de sequences e generator's nativos;
  - Deve ser configurado utilizando o nome completo de uma subclasse de "net.sf.hibernate.dialect.Dialect"
    - Exemplo: hibernate.dialect=net.sf.hibernate.dialect.PostgreSQLDialect

Configuração do Hibernate

- Logging
  - O Hibernate utiliza o Apache Commons-logging, o qual redireciona a saída através do log4j ou da API de logging do JDK 1.4;
  - Para utilizar o log4j é necessário que os arquivos “log4j.jar” e “log4j.properties” sejam acessíveis através do classpath da aplicação;

FPSW-Java



Mapeamento de classes utilizando Hibernate

- Os mapeamentos são declarados em um ou vários arquivos *.xml (Hibernate mapping files);
- Principais regras para classes persistentes:
  - Devem possuir métodos “get” e “set” para os atributos persistentes;
  - Devem possuir um construtor padrão (implícito ou explícito) para que o Hibernate possa instanciar classes através do método “newInstance()”;
  - Devem possuir um atributo identificador (não obrigatório para “Dependent objects”), de forma a possibilitar “cascade updates” e chamada de método saveOrUpdate (inteligente);
  - Não devem ser “final” de forma a possibilitar o uso de proxies;

Mapeamento de classes utilizando Hibernate

- Principais elementos dos Hibernate mapping files:
  - **<class>** - Define a classe persistente e a tabela do BD;
  - **<id>** - Define o atributo que será o identificador da instância da classe e o tipo de generator que será utilizado pelo atributo id;
  - **<composite-id>** - Utilizado para mapear classes para tabelas que possuem chaves compostas;
  - **<property>** - Define o mapeamento de um atributo persistente, caso o atributo já tenha sido definido nos elementos <id> ou <composite-id> ocorrerá um erro;
  - **<many-to-one>** - Define o lado “one” um relacionamento “um para muitos”;
  - **<one-to-one>** - Define um lado “one” de um relacionamento “um para um”;

Mapeamento de classes utilizando Hibernate

- **<component>** - Define o mapeamento dos atributos de uma classe dependente para colunas da tabela mapeada para a classe pai;
- **<dynamic-component>** - Define que um Map seja mapeado como um componente onde as propriedades do componente serão as chaves do elementos do Map;
- **<subclass>** - Define o mapeamento dos atributos de uma subclasse para colunas da tabela mapeada para a classe pai. É necessário definir um atributo “discriminador” na classe pai, além de um valor discriminante para cada classe da hierarquia. Persistência Polimórfica;

Mapeamento de classes utilizando Hibernate

- **<joined-subclass>** - Define o mapeamento dos atributos de uma subclasse para colunas de uma tabela mapeada para a subclasse. É necessário definir um elemento chave <key>, o qual irá apontar para a chave estrangeira da tabela que foi mapeada para a classe pai. Não é necessário definir um elemento “discriminador”;

Mapeamento de classes utilizando Hibernate

- Tipos do Hibernate
- Tipos básicos:
  - integer, long, short, float, double, character, byte, boolean, yes_no, true_false – Tipos primitivos Java ou Wrapper classes são mapeados para os correspondentes tipos de dados do SQL;
  - string – mapeado para tipos de dados VARCHAR;
  - date, time, timestamp - java.util.Date são mapeados para os tipos de dados SQL (DATE, TIME e TIMESTAMP);
  - calendar, calendar_date - java.util.Calendar são mapeados para os tipos de dados SQL (TIMESTAMP e DATE);

Mapeamento de classes utilizando Hibernate

- big_decimal - java.math.BigDecimal para o tipo de dados SQL NUMERIC ou equivalente.
- locale, timezone, currency - java.util.Locale, java.util.TimeZone e java.util.Currency para o tipo de dados SQL VARCHAR.
- class - java.lang.Class para o tipo de dados SQL VARCHAR. A Classe é mapeada utilizando o seu nome completo.
- binary - byte arrays para o tipo binário SQL apropriado.
- text – Strings Java strings para o tipo de dados SQL CLOB ou TEXT.
- serializable – Classes javas serializáveis para o tipo binário SQL apropriado.

Mapeamento de classes utilizando Hibernate

- Tipos enumerados persistentes
  - Implementar a interface "net.sf.hibernate.PersistentEnum";
- Tipos definidos pelo usuário
  - Implementar a interface "net.sf.hibernate.UserType" ou "net.sf.hibernate.CompositeUserType";
- Os tipos declarados acima necessitam que o seu nome completo seja informado no atributo "type" do elemento <property>;

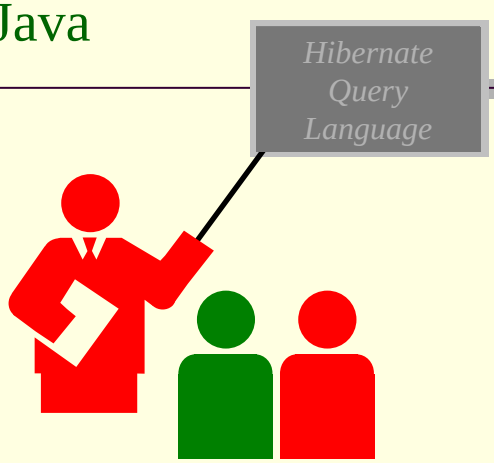
Mapeamento de Coleções utilizando Hibernate

- Pode persistir instâncias de java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List;
- O Hibernate irá trocar as instâncias de Map, Set e List pelas suas implementações. A principal razão desta estratégia é o suporte a "Lazy instantiation";
- <set>, <list>, <map>, <bag>, <array> e <primitive-array> são os elementos que podem ser mapeados pelo Hibernate;

Mapeamento de Coleções utilizando Hibernate

- A Java Collections Framework não possui uma interface Bag. Uma “bag” seria uma coleção de elementos não-indexados e não-ordenados que podem repetir o mesmo elemento inúmeras vezes. O Hibernate permite que uma bag seja implementada através de uma “property” List ou Collection;
- Os elementos de uma coleção podem ser mapeados das seguintes maneiras: <element>, <composite-element>, <one-to-many>, <many-to-many>;

FPSW-Java



Hibernate Query Language (HQL)

- Polimórfica
  - Ex: “from java.lang.Object o” retorna todos os objetos persistentes;
  - “from hibtest.ClienteDO cli where cli.class = hibtest.ClientePJDO”- Retorna todos os clientes mapeados para a subclasse “hibtest.ClientePJDO”
- Funções de Agregação: avg(...), sum(...), min(...), max(...), count(*), count(...), count(distinct ...)

Hibernate Query Language (HQL)

- Cláusula SELECT
  - “select elements(f.produtos) from hibtest.FornecedorDO f” – Retorna as coleções de produtos associadas aos fornecedores retornados, através da função “elements”;
- Cláusula WHERE
  - Possui várias expressões e funções úteis;
  - “from hibtest.FornecedorDO f where size(f.produtos) > 10” – Retorna os fornecedores que possuem mais de 10 produtos associados;
- Suporte a “group by”;
- Suporte a subquery’s;
- Suporte a “Named parameters”;

Ferramentas que acompanham o Hibernate (visão geral)

- SchemaExport
  - Geração de “schema” de banco de dados a partir dos arquivos de mapeamento do Hibernate;
  - Possibilita atualizações incrementais;
- CodeGenerator
  - Geração de código a partir dos aquivos de mapeamento do Hibernate;
  - Possibilita a criação de classes de busca (Finder classes);
- MapGenerator
  - Geração dos arquivos de mapeamento do Hibernate a partir das classes persistentes;
  - Melhor utilizar o XDoclet;

FPSW-Java

Exemplo de Aplicação com Hibernate

Arquitetura e Design da Aplicação

- O cenário da aplicação
  - Modelo Relacional
    - Existente, Inteiramente novo ou parcialmente novo
  - Configuração
    - Servidor de aplicação, Spring, Properties, XML, etc.
  - Transação
    - Controle transacional JDBC ou JTA; nível read-commited, etc.
- Data Access Object - DAO
  - O pattern DAO (Data Access Object) consiste na separação do código relacionado ao acesso dos dados, do código de negócio.
  - Utilizando DAO com Hibernate podemos deixar transparente o uso do Hibernate para o código de negócio.
- Uso de Data Transfer Objects – DTO
  - Com Hibernate não existe a necessidade de trabalhar com DTOs

Implementação

- Classes persistentes
  - Entidade, Herança e Relacionamentos
- Transação
  - HibernateFilter
- Classes Utilitárias
  - HibernateUtil
- Data Access Object
  - DAOFactory
  - ProjetoDAO e ProjetoDAOHibernate, RegistroDAOHibernate

Implementação e Ambiente Desenvolvimento

- Negócio
  - Registro.close()
  - BugTrackerManager
- Controller e Apresentação
  - BugTrackerServlet e JSPs
- Automatizar o maior número possível de tarefas
- Ambiente de teste que dependa o mínimo possível de recursos externos ao sistema
- Automatizando tarefas com Ant
  - Compilação, Arquivo jar e war
  - Gerando o mapeamento com Xdoclet
  - Gerando o script do banco de dados

Ambiente de Desenvolvimento

- Ambiente de testes
  - Mock Objects: Implementações falsas de recursos que são utilizadas em rotinas de testes, como por exemplo da API JDBC
  - Junit e HSQL: Pelo fato de Hibernate oferecer transparência, podemos utilizar um banco de dados em memória para servir de recurso de nossas rotinas de teste. Independente de qual será o banco de dados de produção.

Desenvolvimento

- Uma explicação simples sobre o desenvolvimento no Hibernate:
  - Crie as tabelas do BD
  - Crie um bean representando o objeto codificado.
  - Crie um arquivo de mapeamento para que o Hibernate saiba quais atributos correspondem aos campos SQL.
  - Crie um arquivo de configuração informando o Hibernate sobre as configurações do BD.
  - Comece a utilizar a API do Hibernate
- Existem ferramentas que auxiliam na geração de Beans a partir de SQL ou o contrário (e até plug-ins que criam o arquivo de mapeamento automaticamente).

Exemplo

1o Passo: Criar a tabela

Para simplificar vamos usar apenas uma tabela de usuários.

```
CREATE TABLE usuario (
 Login varchar(20) NOT NULL default '0',
 Nome varchar(40) default NULL,
 Senha varchar(20) default NULL,
 Email varchar(40) default NULL,
 UltimoAcesso datetime default NULL,

 PRIMARY KEY (Login)
);
```

Exemplo (cont.)

2o Passo: Escrever o Bean

O Hibernate funciona através de métodos get/set dos objetos, e por isso é necessário cria-los.

```
package com.ismael.demo;
import java.util.Date;
public class Usuario implements java.io.Serializable {
  private String ID;
  private String nome;
  private String senha;
  private String email;
  private Date ultimoAcesso;
  public Usuario()  { }
```


Exemplo (cont.)

```
  public String getID()  { return ID; }
  public void setID(String ID)  { this. ID = ID; }
  public String getNome()  { return nome; }
  public void setNome (String nome)  { this.nome = nome; }
  public String getSenha()  { return senha;  }
  public void setSenha(String senha){ this.senha = senha; }
  public String getEmail ()  { return email;}
  public void setEmail (String email){ this.email = email;}
  public Date getUltimoAcesso() { return ultimoAcesso; }
  public void setUltimoAcesso (Date ultimoAcesso)  {
                      this.ultimoAcesso = ultimoAcesso; }
}
```

Exemplo (cont.)

3o Passo: Escrever o arquivo de mapeamento

Essa é a parte onde se diz ao Hibernate a correspondência entre os atributos do objeto e os campos na tabela. E para tanto o padrão utilizado é XML.

A maneira mais limpa (e por conseqüência, de fácil manuntenção) é escrever um arquivo de mapeamento para cada objeto. Caso o nome do arquivo seja o mesmo do objeto e esteja localizado no mesmo diretório então fica ainda mais fácil. A seguir um exemplo do arquivo Usuario.hbm.xml:

Exemplo (cont.)

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
    2.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
 <class name="com.ismael.demo.Usuario" table="usuario">
    <id name="ID" column="Login" type="string">
      <generator class="assigned"/>
    </id>
    <property name="Nome" column="Nome" type="string"/>
    <property name="senha" column="Senha" type="string"/>
    <property name="email" column="Email" type="string"/>
    <property name="ultimoAcesso" column="UltimoAcesso" type="date"/>
 </class>
</hibernate-mapping>
```

Exemplo (cont.)

- A primeira marcação interessante é a “class”. Ela diz qual classe corresponde a qual tabela no BD.
- A marcação “ID” corresponde à chave primária da tabela. “Nome” mapeia o atributo nome ao campo da tabela (column). “Generator” diz como o Hibernate deve gerar a chave, no exemplo “assigned” faz com que a aplicação gere o identificador antes que o método save() seja chamado.
- As outras marcações funcionam de forma similar apenas fazendo a correspondência entre objeto e tabela.

Exemplo (cont.)

4o Passo: Escrever o arquivo de configuração

A melhor maneira de dizer ao Hibernate onde achar o BD é alimenta-lo com um objeto de configuração contendo os endereços para conexão, senhas, usuário etc. Nomeando o arquivo como hibernate.properties e colocando-o em seu classpath o Hibernate o usará automaticamente.

```
hibernate.dialect=net.sf.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://127.0.0.1:3306/
hibernate.connection.username=user
hibernate.connection.password=pwd
```

Exemplo (cont.)

- O exemplo usa um driver mysql. Existem ainda outras propriedades que podem ser usadas para otimizar o acesso ao BD. O pacote do Hibernate já vem com um arquivo de exemplo padrão que o usuário pode apenas adaptar para seu próprio uso.
- Um arquivo XML também pode ser utilizado (hibernate.cfg.xml). Ele sempre terá preferência sobre o arquivo hibernate.properties e deverá estar localizado na raiz do classpath.

Exemplo (cont.)

5o Passo: Comece a utilizar a API do Hibernate

Nesse ponto temos:

- **Uma tabela de usuários no BD**
- **Usuario.java – o objeto com o qual desejamos trabalhar**
- **Usuario.hbm.xml – o arquivo de mapeamento do Hibernate**
- **hibernate.properties – arquivo de configuração para a conexão com BD**

Para usar o Hibernate no código fonte:

A. Crie um objeto de configuração
B. Informe a configuração sobre o tipo de objeto que se deseja mapear
C. Crie uma sessão com o BD
D. Carregue, armazene, apague ou consulte suas instâncias
E. Use flush() para confirmar as mudanças feitas no BD

Utilização

Passos A e B

O objeto de configuração é baseado no arquivo de configuração (hibernate.properties).

```
Configuration config = new Configuration();
Config.addClass(Usuario.class);
```


Utilização (Cont.)

Passo C

O objeto de sessão representa a conexão com o BD. Poderia ser informado ao Hibernate a cada pedido mas é mais fácil utilizar o arquivo de configuração feito no passo 4.

```
SessionFactory sf = config.buildSessionFactory();
Session session = sf.openSession();
```

Esses comandos podem ser utilizados pois o hibernate.properties está sendo utilizado e foi colocado no classpath. Fosse de outra maneira então as configurações deveriam ser parâmetros de buildSessionFactory().

Utilização (Cont.)

Passo D

Podemos usar os objetos normalmente. Esse é um exemplo para a gravação de um novo usuário:

```
Usuario newUser = new Usuario();
newUser.setID("brunor");
newUser.setNome("Bruno Ribeiro");
newUser.setSenha ("1234");
newUser.setEmail (“blribeiro5696@globo.com”);
newUser.setUltimoAcesso(new Date());
// Chamada do Hibernate para gravar usuário
session.save(newUser);
```

Percebe-se que a grande vantagem do Hibernate é que não é preciso adicionar muito código. Basta uma chamada ao Hibernate quando tudo estiver terminado.

Utilização (Cont.)

Passo E

De tempos em tempos o objeto de sessão do Hibernate deve sincronizar as cópias dos objetos armazenadas em sua memória com o BD. Para isso é usado o comando flush():

```
session.flush();
session.close();
```

Exemplo

```
package com.ismael.demo;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import com.ismael.demo.Usuario;
import java.util.Date;

public class UsuarioTest {
 public static void main(String[] args) throws Exception{
   // Cria a configuração baseado no arquivo de configuração
   Configuration config = new Configuration();
   // Informa quais as classes mapeadas.
   config.addClass(Usuario.class);
```


Exemplo (cont.)

```
   // Cria uma sessão para a conexão com o BD
   SessionFactory sessionFactory =
                        config.buildSessionFactory();
   Session session = sessionFactory.openSession();
   // Armazena um usuário no BD
   try {
    Usuario newUser = new Usuario();
    newUser.setID("brunor");
    newUser.setNome("Bruno Ribeiro");
    newUser.setSenha ("1234");
    newUser.setEmail (“blribeiro5696@globo.com”);
    newUser.setUltimoAcesso(new Date());

    // Chamada do Hibernate para gravar usuário
    session.save(newUser);
   }
```

Exemplo (cont.)

```
   finally  {
      // Feche a sessão
      session.flush();
      session.close();
   }
   sessionFactory.close();
  }
}
```