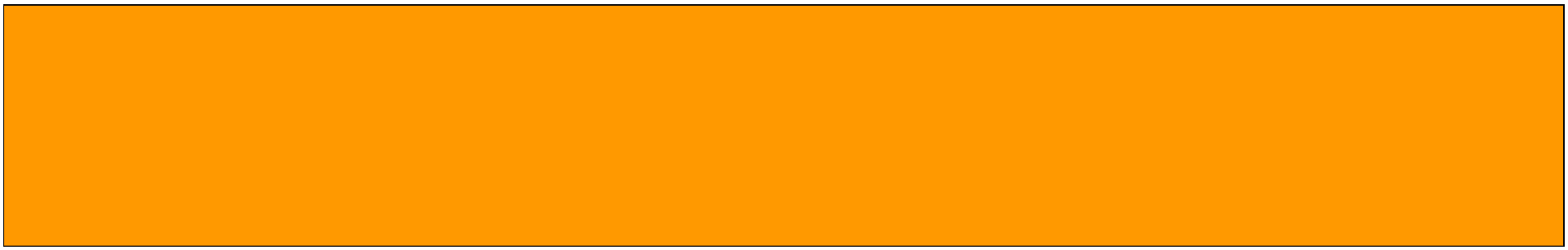
Hibernate

Uma visão geral sobre o framework padrão *de fato* para mapeamento objeto-relacional



Sobre o autor

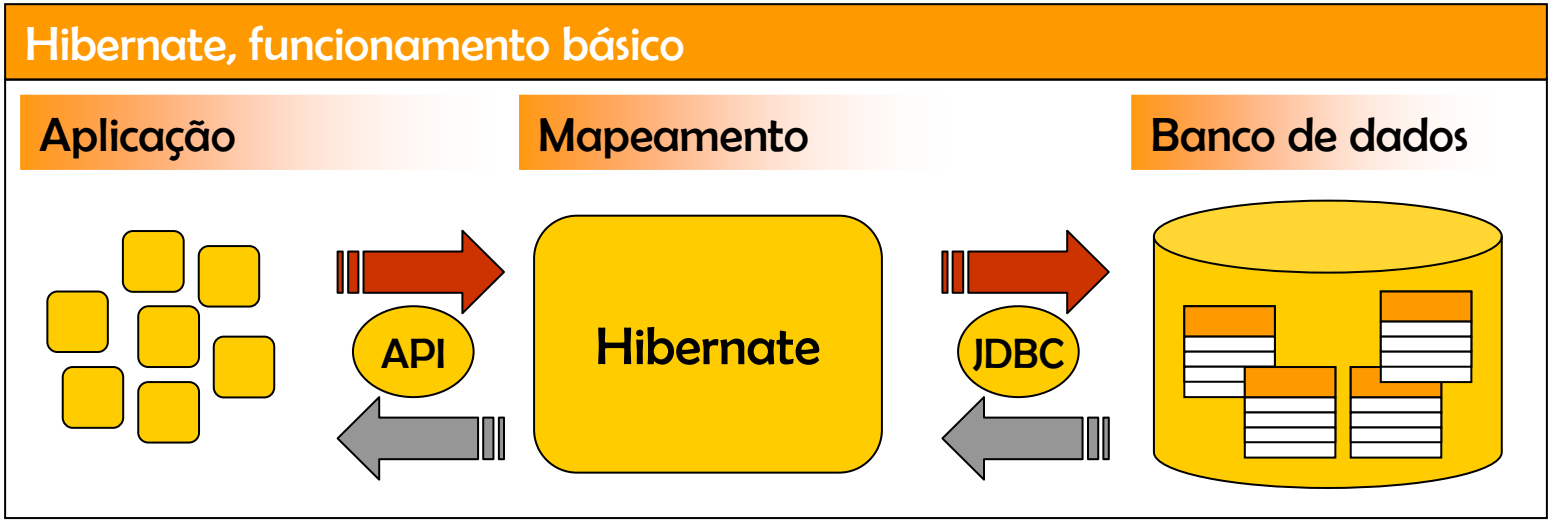
- Marcelo Mrack, mmrack@gmail.com
  - 29 anos, 8 em TI, 6 em Java
  - Bacharel em C. Computação, UNISC – 2001
  - Mestrando em C. Computação, UFRGS – 2006
  - Atuação em projetos web e desktop n camadas
  - Sócio e arquiteto na 3Layer Tecnologia
  - Projetista na CWI Software
  - Consultor e instrutor Hibernate, Java EE
  - Especialidades: IHC, Patterns, geradores, PU Ágil e UML
  - http://merlin.dev.java.net
  - http://telasdecadastro.blogspot.com

Sumário

- Visão geral
- Características gerais
- Arquitetura
- Funcionamento
- Associações, coleções e herança
- Cache
- HQL
- Outras características
- Ferramentas e utilitários
- Dicas
- Comentários finais

Visão geral

- O Hibernate é um framework de mapeamento objeto-relacional para a linguagem Java
  - Conjunto de classes, interfaces e configuração que permite simplificar o trabalho de persistir e recuperar objetos Java em banco de dados relacionais

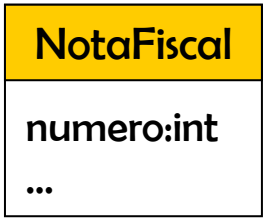


Degustação

- Um exemplo simples de uso do Hibernate

Persistência...

```
1  NotaFiscal nf = new NotaFiscal();
2  nf.setNumero(numero);
3  //...outros setters...
4
5  Session session = HibernateUtil.
6    getSessionFactory().getCurrentSession();
7
8  session.beginTransaction();
9  session.save(nf);
10 session.getTransaction().commit();
```

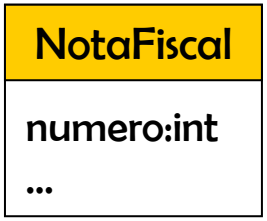


Degustação

- Um exemplo simples de uso do Hibernate

Persistência...

```
1  NotaFiscal nf = new NotaFiscal();
2  nf.setNumero(numero);
3  //...outros setters...
4
5  Session session = HibernateUtil.
6    getSessionFactory().getCurrentSession();
7
8  session.beginTransaction();
9  session.save(nf);
10 session.getTransaction().commit();
```



Recuperação...

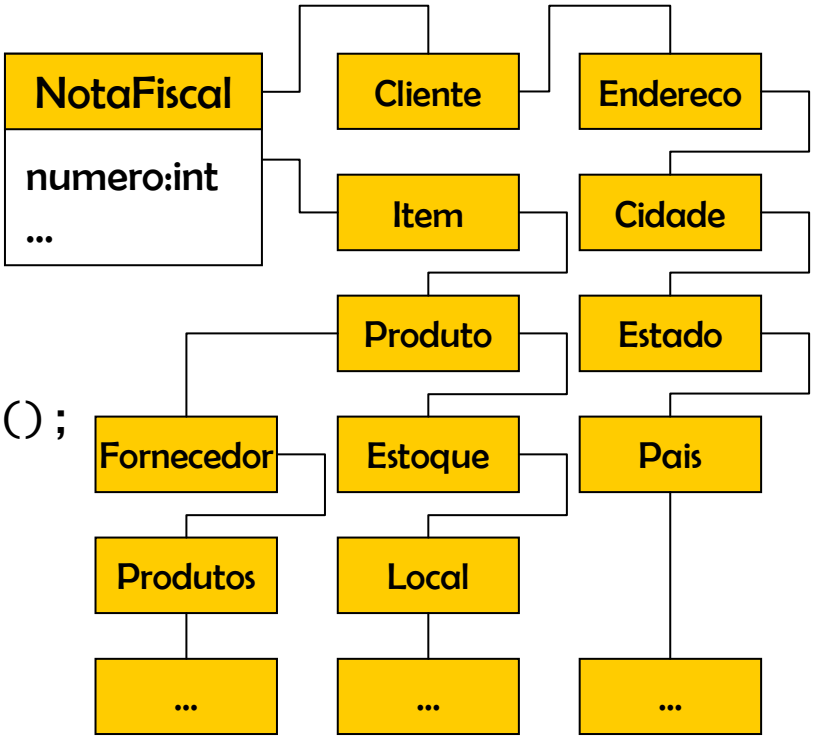
```
1  Session session = HibernateUtil.
2    getSessionFactory().getCurrentSession();
3  NotaFiscal nf = (NotaFiscal) session.load(NotaFiscal.class, 1001);
```

Degustação

- Um exemplo simples de uso do Hibernate

Persistência...

```
1 NotaFiscal nf = new NotaFiscal();
2 nf.setNumero(numero);
3 //...outros setters...
4
5 Session session = HibernateUtil.
6   getSessionFactory().getCurrentSession();
7
8 session.beginTransaction();
9 session.save(nf);
10 session.getTransaction().commit();
```



Recuperação...

```
1 Session session = HibernateUtil.
2   getSessionFactory().getCurrentSession();
3 NotaFiscal nf = (NotaFiscal) session.load(NotaFiscal.class, 1001);
```

Histórico

- O início
  - Concepção no final de 2001
  - Projeto pessoal, de Gavin King na Cirrus Technologies, AU
  - Descontentamento com o modelo J2EE CMP
- A evolução
  - jul/2002 - versão 1.0
    - inicialização tardia
  - jan/2003 – versão 2.0
    - dialeto Oracle 9
  - jul/2004 – versão 3.0
    - net.sf.hibernate > org.hibernate
    - adequação EJB3 (JSR220)
  - nov/2006 – versão 3.2.1
    - release corrente

Características gerais

- Abordagem totalmente OO
- Suporte à mais de 20 SGBD
- Gera comandos SQL nativos para cada SGBD
- Suporte total ao Java 1.5
- Opera em ambientes *standalone* e sob *containers*

Características gerais

- Alta Performance
  - 2 Níveis de Cache
  - SQL Nativo
  - Comandos pré-compilados
  - Queries nativas com mapeamento automático
- Suporte à transações
  - Standalone, demarcadas explicitamente
  - Gerenciada por container (XA-Transactions), implícitas

Características gerais

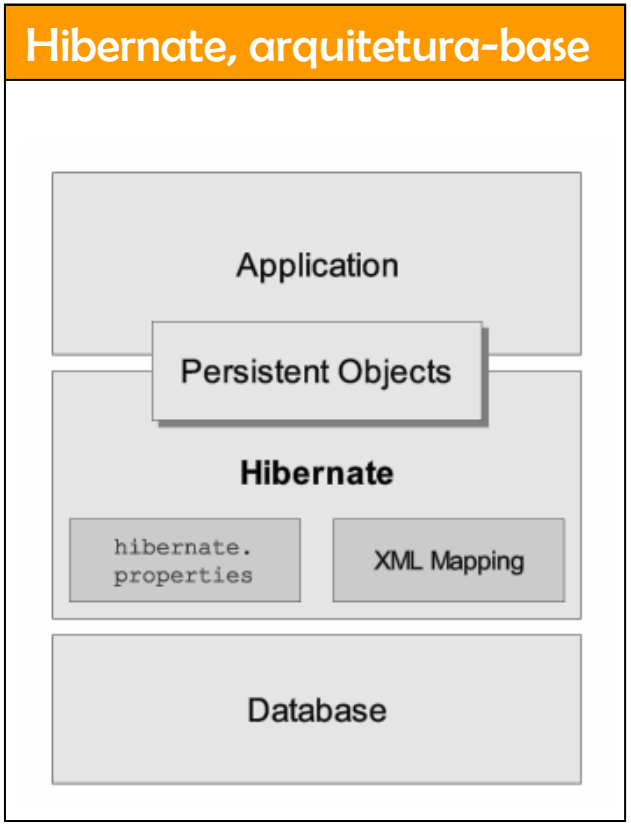
- Linguagem própria de consulta
  - HQL
    - Semelhante ao SQL
    - Orientada a Objeto
    - Muitas funcionalidades embutidas
- Configuração flexível
  - XML
  - Texto puro (arquivo `.properties`)

Características gerais

- Ferramentas e utilitários disponíveis
  - Utilitários
    - Geração/atualização da BD
    - Validação da BD
  - Plugins para IDEs
    - Operação visual
    - Engenharia reversa (geração das classes Java a partir da BD)
- Software livre
- Grande comunidade
- Apoiado pela JBoss (RedHad)

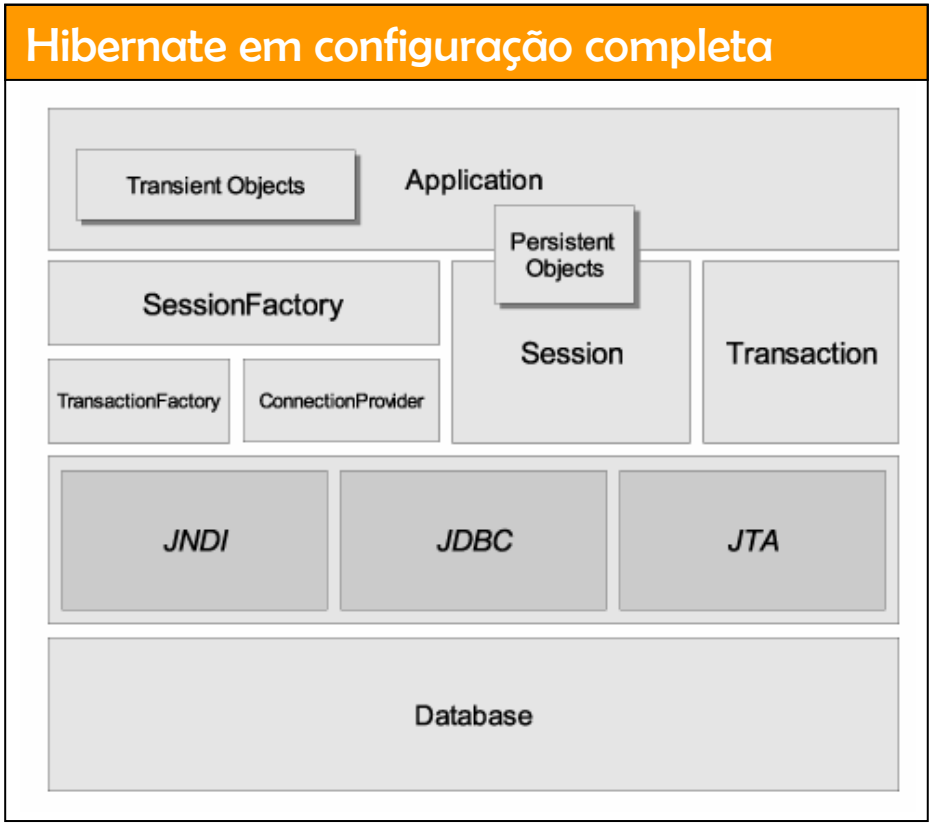
Arquitetura

- Arquitetura-base do Hibernate



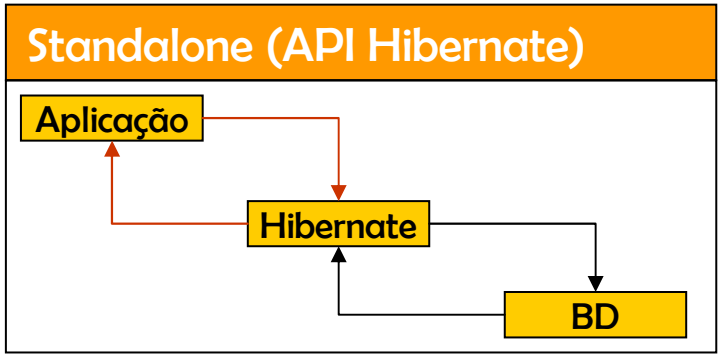
Arquitetura

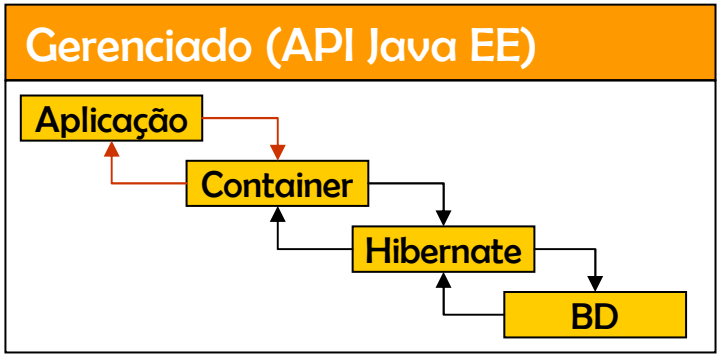
- A configuração completa



Modos de operação

- São dois os modos de operação do Hibernate
  - Standalone
    - Comum para sistemas 2 camadas (desktop ou web). Nele, o Hibernate controla todo o escopo de operação, e a aplicação cliente tem domínio completo da execução do sistema
  - Gerenciado
    - Comum para sistemas n camadas. Nele, o Hibernate é configurado como um serviço no Servidor de Aplicação, e a aplicação cliente solicita serviços do framework





Funcionamento

- Em suma, o funcionamento do Hibernate é:
  1. Para cada objeto do sistema, existe uma configuração de mapeamento, na forma:
     - classe <> tabela
     - propriedade <> coluna | relacionamento
  2. Para cada base de dados, existe um arquivo de configuração que define:
     - Parâmetros de conexão
     - Pool
     - Comportamento padrão
     - etc.
  3. Para persistir e recuperar objetos, a aplicação utiliza sua API

Modelo de desenvolvimento

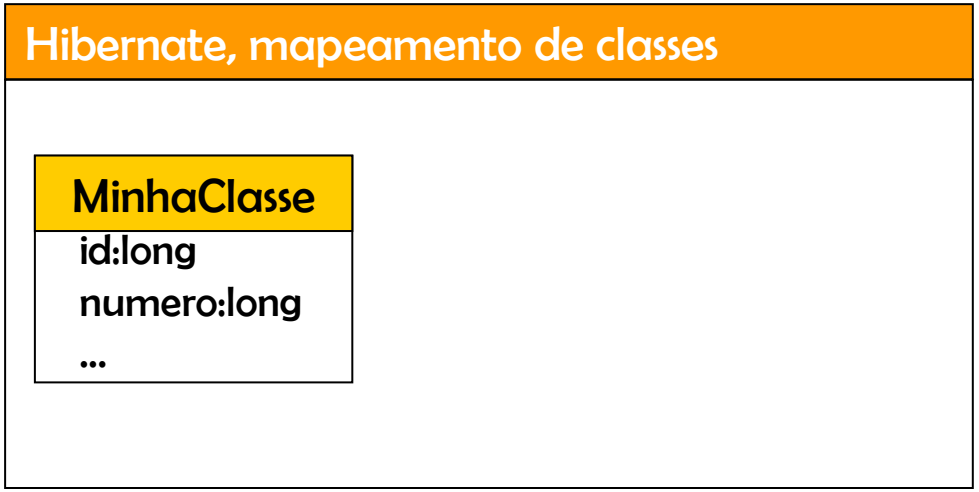
- O processo de desenvolvimento com o Hibernate pode ser de duas formas:
  - TopDown: Modela-se OO e gera-se a base
    - Modela-se as classes do sistema
    - Cria-se os arquivos de mapeamento e configuração
    - Gera-se a base de dados
    - Usa-se a API para persistir e recuperar objetos
  - BottomUp: Dada uma BD existente, geram-se os artefatos
    - Efetua-se a engenharia reversa de uma BD via plugin Hibernate Tools, gerando-se:
      - classes Java, arquivos de mapeamento e configuração
    - Ajusta-se o mapeamento, as classes e a configuração (se necessário)
    - Usa-se a API para persistir e recuperar objetos

Classes de persistência

- O Hibernate não força nenhuma regra sobre os objetos a serem persistidos, porém, algumas práticas são "fortemente" recomendadas:
  - Seguir o modelo POJO, onde:
    - Construtor vazio, no mínimo com visibilidade de pacote
    - Métodos getters/setters públicos padrões
      - (nada além de `return this.x` e `this.x = x`)
    - Fields não públicos
  - Providenciar um atributo identificador (ID)
  - Não usar classes `final`

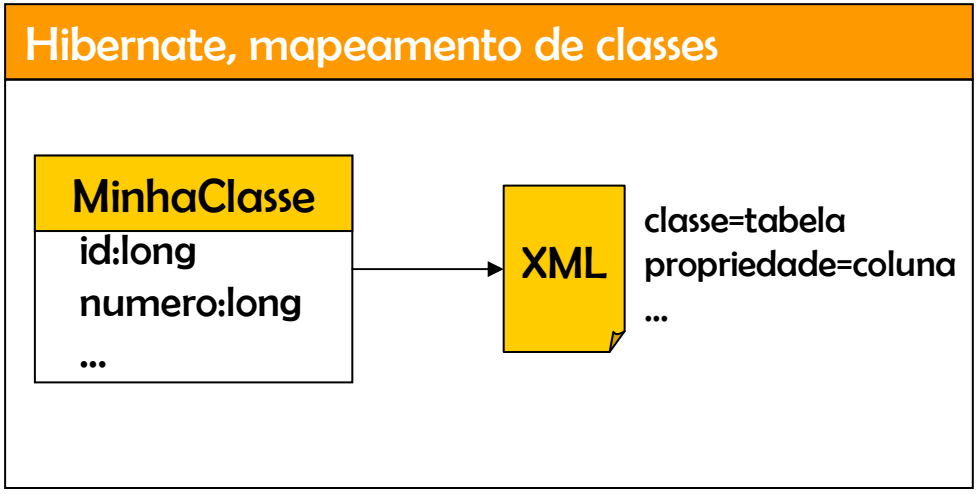
O processo de mapeamento

- A configuração do mapeamento OO-ER é feita por classe, através de:
  - XML
  - Anotações (JSR175) (exige pacote extra)



O processo de mapeamento

- A configuração do mapeamento OO-ER é feita por classe, através de:
  - XML
  - Anotações (JSR175) (exige pacote extra)



Usando XML

```
<class name="CreditCardPayment" table="CR
    <id name="id" type="long" column="CRE
        <generator class="native"/>
    </id>
    <discriminator column="CREDIT_CARD" t
    <property name="amount" column="CREDI
    ...
    <subclass name="MasterCardPayment" di
    <subclass name="VisaPayment" discrimi
```

O processo de mapeamento

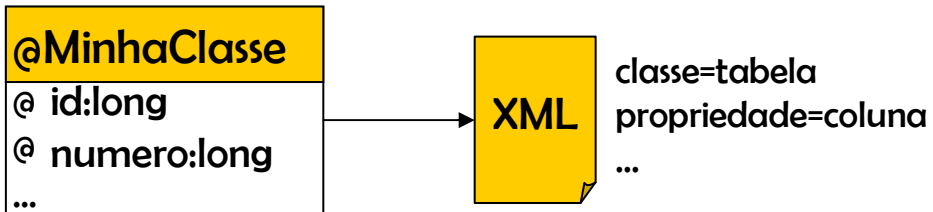
- A configuração do mapeamento OO-ER é feita por classe, através de:
  - XML
  - Anotações (JSR175) (exige pacote extra)

Usando anotações

```
@Entity
@Inheritance(strategy = Inherita
@javax.persistence.NamedQueries(
    @NamedQuery( name="documento
    @NamedQuery( name="documento
)
public abstract class Documento

  private String numero;
```

Hibernate, mapeamento de classes



Usando XML

```
<class name="CreditCardPayment" table="CR
    <id name="id" type="long" column="CRE
        <generator class="native"/>
    </id>
    <discriminator column="CREDIT_CARD" t
    <property name="amount" column="CREDI
    ...
    <subclass name="MasterCardPayment" di
    <subclass name="VisaPayment" discrimi
```

O Hibernate e os IDs

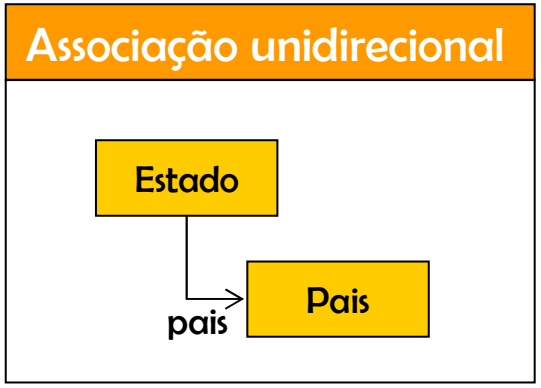
- Uma vez que objetos são recomendados a terem um ID, o Hibernate encarrega-se de gerenciá-los
- Para isso são usados algoritmos de geração:
  - identity
  - native
  - increment
  - uuid
  - etc.
- Cada algoritmo tem uma aplicação específica
  - increment não deve ser usado em cluster, por exemplo

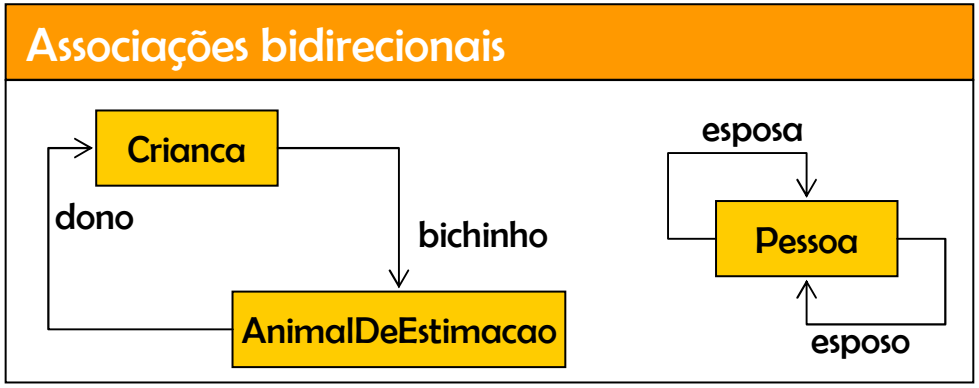
O modelo OO e o Hibernate

- O Hibernate suporta completamente o modelo OO do Java, incluindo
  - Associações simples e bidirecionais
  - Coleções
  - Herança
  - Interfaces
  - Tipos de dados nativos e wrappers
  - Objetos simples e compostos
  - Atributos estáticos
  - Enumerações
  - Qualquer tipo definido pelo usuário

Associações

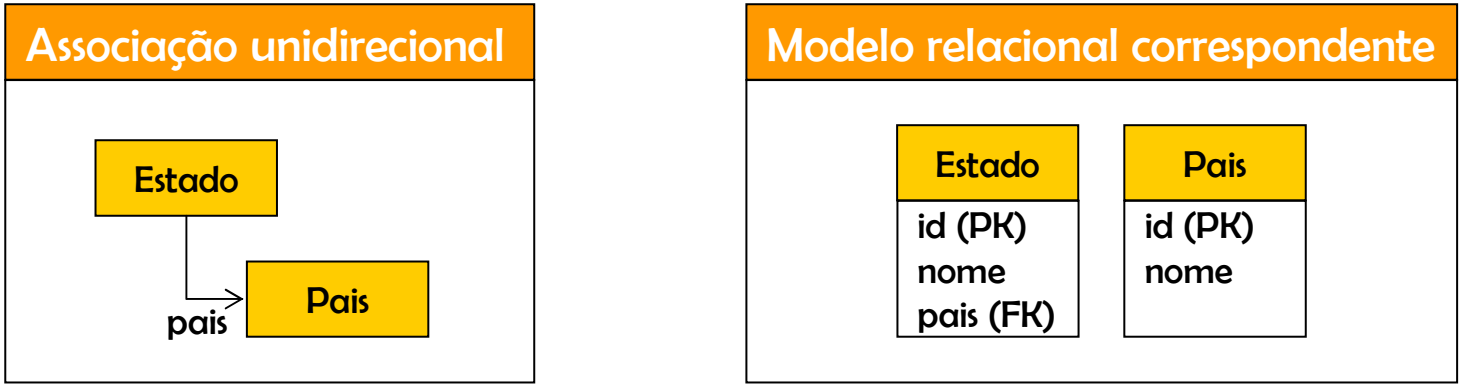
- Unidirecionais
  - A referencia B
- Bidirecionais
  - A referencia B que referencia A





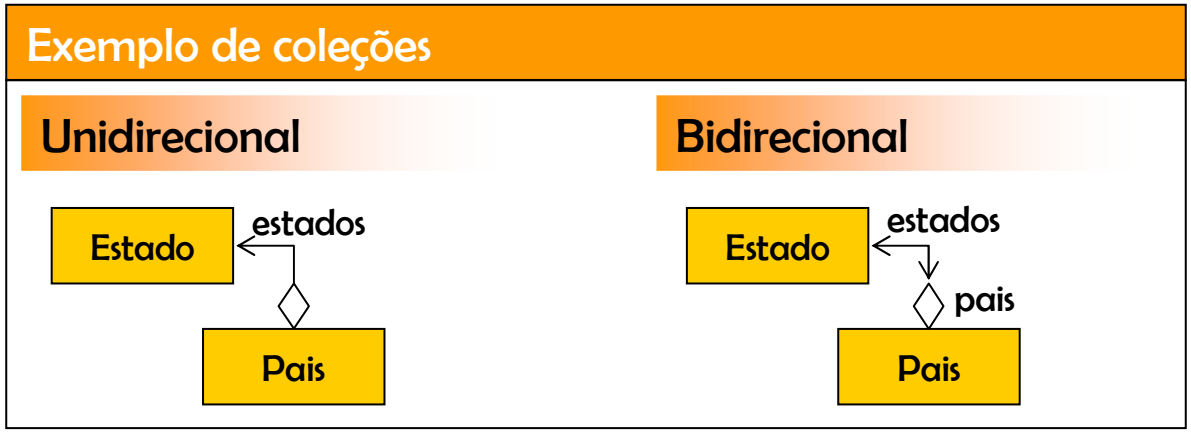
Associações e o modelo relacional

- Associações são mapeadas para o modelo relacional através de um chave estrangeira na tabela equivalente à classe que declara a associação



Coleções

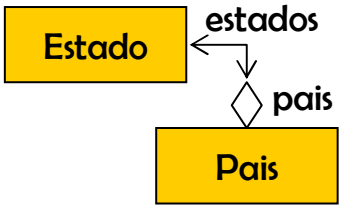
- Um objeto A referencia uma coleção de objetos B



Coleções e o modelo relacional

- Coleções podem ser mapeadas através de chaves estrangeiras, ou via tabelas intermediárias

Coleção bidirecional



Possíveis mapeamentos

Via chave estrangeira

Estado	Pais
id (PK)	id (PK)
nome	nome
pais (FK)	

Via tabela associativa

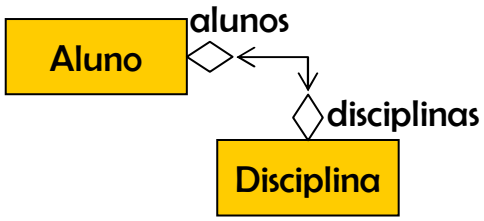
Estado	PaisEstado	Pais
id (PK)	paisId (FK)	id (PK)
nome	estadoId (FK)	nome

Relacionamentos múltiplos (n-n)

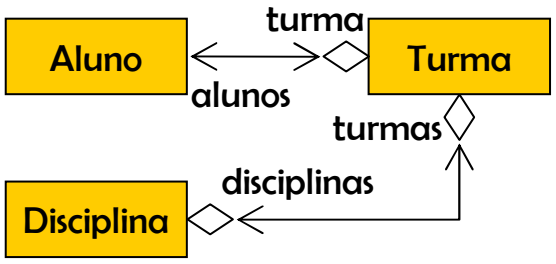
- Muitas vezes, os relacionamentos entre as classes podem ter cardinalidade maior que 1, nesse caso:

Modelo OO

Sem informações por item



Com informações por item



Modelo ER

Sem informações por item

Aluno	AlunoDisciplina	Disciplina
id (PK) nome	alunoId (FK) disciplinaId (FK)	id (PK) nome

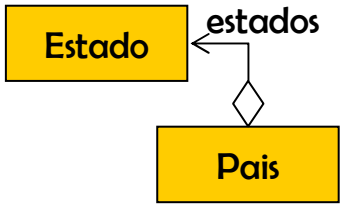
Com informações por item

Aluno	Turma	Disciplina
id (PK) nome	nome (PK) alunoId (FK) disciplinaId (FK) horario	id (PK) nome

Carregamento de dependências

- Associações e coleções podem ser carregadas de duas formas:
  - Antecipadamente: Quando o objeto principal é carregado, a associação (ou coleção) também o é.
  - Tardiamente: Quando o objeto principal é carregado, a associação (ou coleção) não o é.

Carregamento de dependências



Antecipado

```
Pais p = (Pais) session.load(Pais.class,123);
//p.getEstados() inicializada
```

Tardio

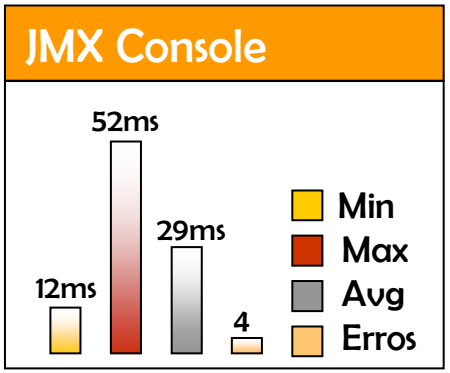
```
Pais p = (Pais) session.load(Pais.class,123);
//p.getEstados() não inicializada
```

Mais sobre coleções

- Coleções podem ser indexadas ou não
  - Coleções indexadas exigem uma coluna a mais (de ordem) na tabela destino
- Coleções podem ser ordenadas explicitamente
  - Durante o mapeamento, é possível especificar uma ordem de busca via ORDER BY
- Coleções podem ter comandos customizados de carregamento
  - O padrão de busca é uma junção entre as tabelas, mas pode ser explicitado um comando diferente

Integração com JMX

- O Hibernate suporta o gerenciamento de serviço através da tecnologia JMX, padrão no Java EE
- Através dela é possível verificar o estado das sessões de trabalho, da fábrica de sessões, estatíticas de cache, log, etc.
- Utilitários visuais podem ser encontrados na web, ou construídos conforme a necessidade



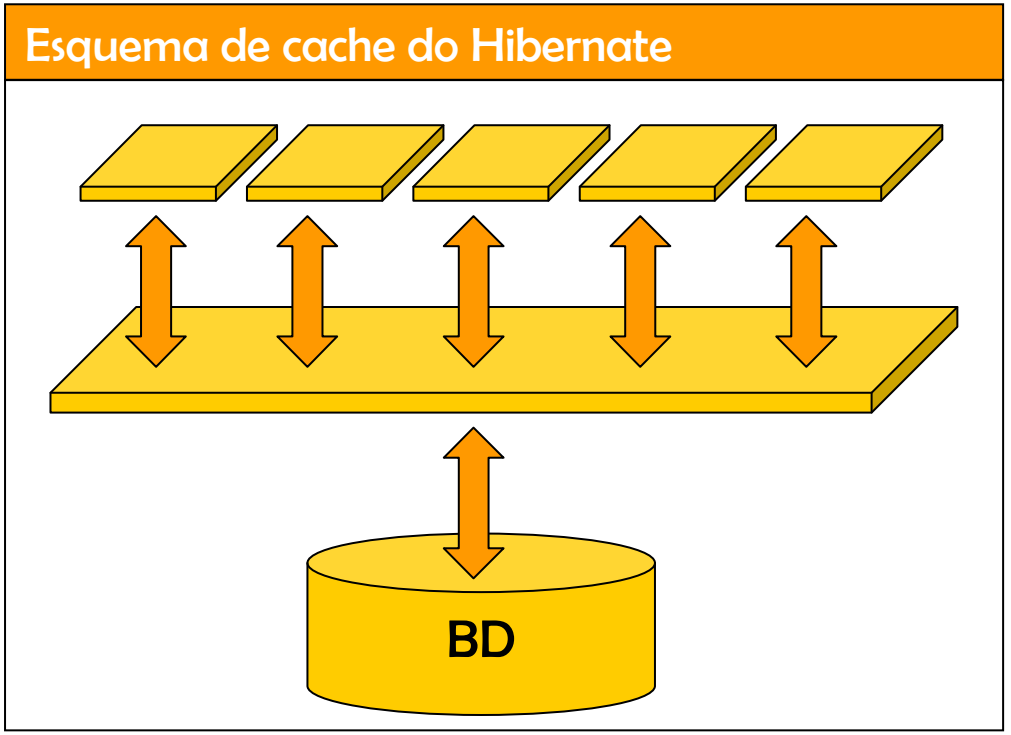
Cache

- O Hibernate possui um mecanismo robusto de cache para os objetos da aplicação, dividido em dois níveis

Cache de primeiro nível
(por sessão)

Cache de segundo nível
(por serviço)

Esquema de cache do Hibernate



Cache de primeiro nível

- O cache de primeiro nível é automático, e seu escopo é a sessão de trabalho
  - Cada objeto persistido ou recuperado durante uma sessão é cacheado e está disponível para todos clientes que acessarem essa mesma sessão
  - Quando a sessão é finalizada, os objetos são liberados

Cache de segundo nível

- O cache de segundo nível é configurável, e seu escopo é variável
  - Objetos e coleções podem ser cacheadas através de várias maneiras, via configuração
  - Diversos engines de cache são suportados:
    - Hashtable, EHCache, SwarmCache, JBoss TreeCache
    - Características diferentes quanto à clusterização, invalidação, suporte à transações, etc.
  - ReadOnly, ReadWrite, nonStrict ReadWrite, Transactional
- Queries podem, explicitamente, abdicar do cache

Cache, um exemplo

- Configuração do cache de segundo nível
  - Pode ser dada por XML, anotações ou por arquivo externo

Configuração básica de um cache usando a engine EhCache

```
<!-- Cache para objetos Lei: 1500 objetos, 1 semana no cache, ~622 bytes * 1500 = ~ 933Kb) -->
<cache name="treelayer.cache.ged.br.com.treelayer.ged.model.beans.Lei"
    maxElementsInMemory="1500"
    eternal="false"
    overflowToDisk="true"
    timeToIdleSeconds="0"
    timeToLiveSeconds="604800"
    diskPersistent="true"
    />
```

Dimensionar os dados no cache é importante, levando em consideração acessos, média de preenchimento dos objetos, memória disponível e tipo de retenção

Usando anotações para especificar parâmetros de cache

```
18 @Entity
19 @javax.persistence.NamedQueries( value={
20     @NamedQuery( name="FolhaDePagamento.buscaFolhasDePagamentoPeloColabor
21     @NamedQuery( name="FolhaDePagamento.buscaTodasAsFolhasDePagamento", (
22   }
23 )
24 @Cache(
25     usage=org.hibernate.annotations.CacheConcurrencyStrategy.READ_WRITE,
26     region="treelayer.cache.ged")
27 public class FolhaDePagamento extends Documento {
28
29   private String colaborador;
```

Transações

- Todas as operações executadas no Hibernate são encapsuladas por transações
  - Standalone, transações demarcadas pelo usuário
  - No container, podem ser automatizadas via JTA pelo Servidor de Aplicação

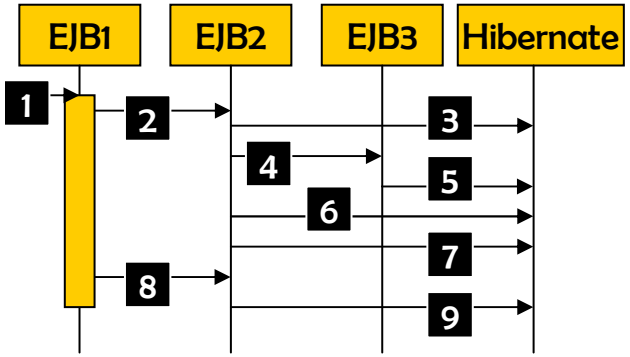
Modo Standalone

1. getSession()
2. beginTransaction()
3. persist()
4. commit()
5. closeSession()

Modo gerenciado

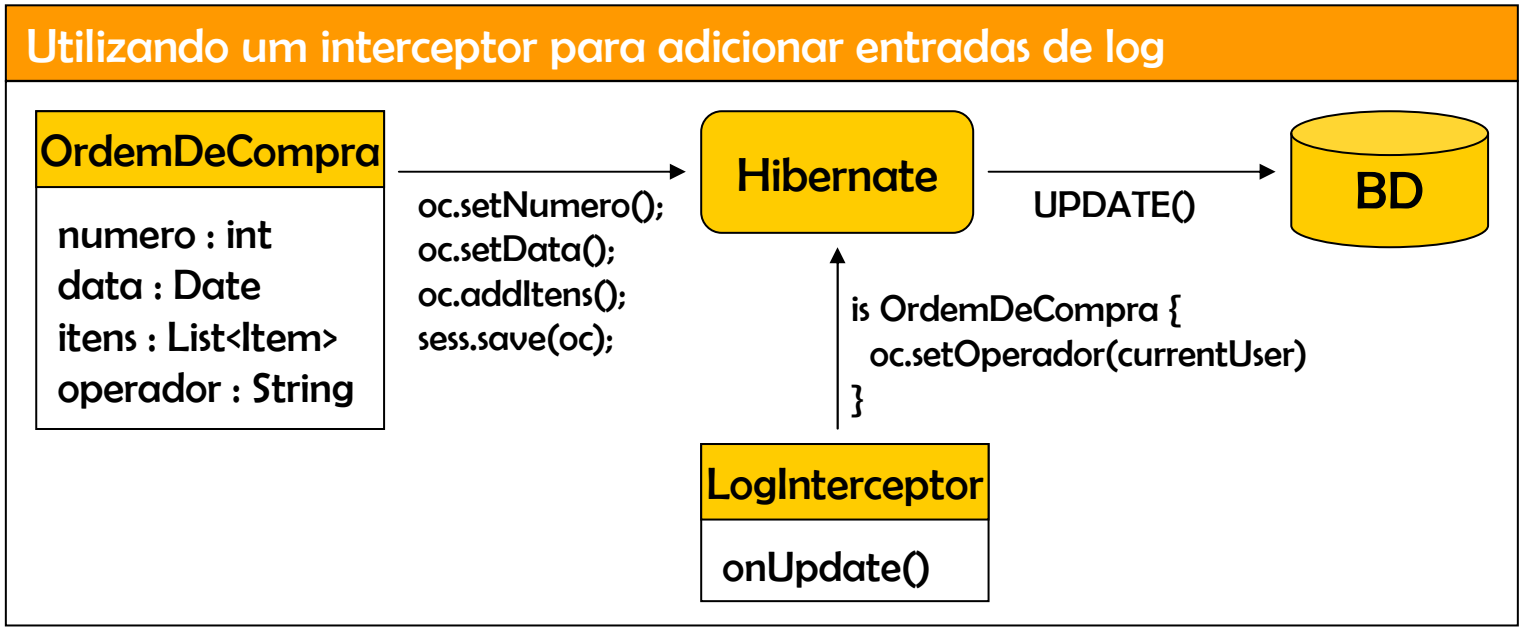
1. getSession()
2. persist()

Chamada típica em container



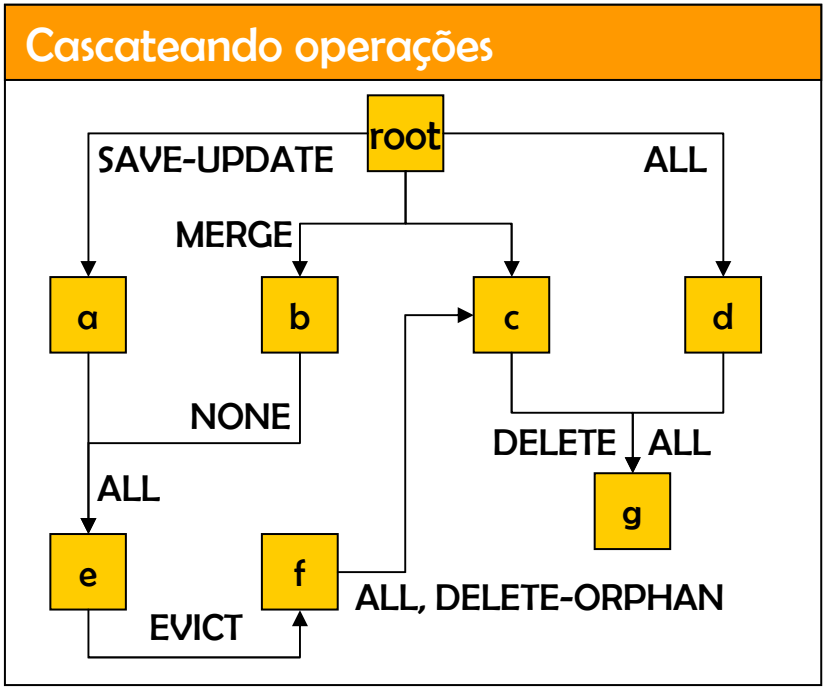
Interceptadores

- São objetos que podem ser anexados à uma sessão de trabalho e executar ações diversas
  - Útil, para execução de operações administrativas, de segurança e auditoria



Cascateamento de operações

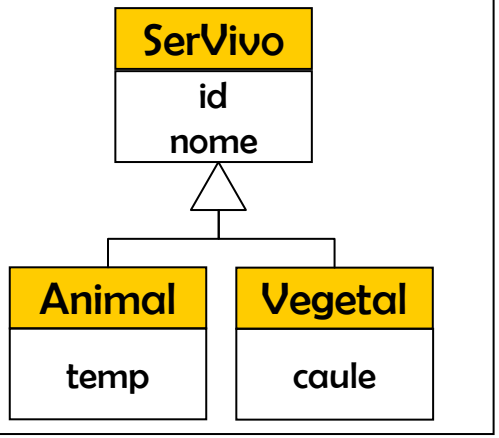
- Por padrão, nenhuma operação de cascateamento é executada. Porém, isso pode ser alterado, conforme as necessidades da aplicação, sendo:
  - Nenhuma
  - Inclusão
  - Atualização
  - Remoção
  - Todas
  - Todas, removendo órfãos
  - Limpeza (de cache)
  - Fusão (em sessão)
  - ou combinações válidas



Herança

- O Hibernate suporta o mapeamento da herança de três formas:
  - Uma tabela por hierarquia
  - Uma tabela por subclasse
  - Uma tabela por classe concreta

Hierarquia de classes



Uma tabela por hierarquia

SerVivo
id | nome | temp | caule | TIPO

Uma tabela por subclasse

SerVivo	Animal	Vegetal
id | nome	id (FK) | temp	id (FK) | caule

Uma tabela por classe concreta

SerVivo	Animal	Vegetal
id | nome	id | nome | temp	id | nome | caule

HQL – Conceito

- O Hibernate possui uma linguagem própria para recuperação dos objetos armazenados na base de dados, é a HQL
- Não segue nenhuma especificação
  - Mas deu origem a uma: a EJB-QL do Java EE
- Projetada para ser simples e poderosa
  - Mais direta que o SQL
  - Menos complexa que a OQL

HQL – Características

- Opera sobre o serviço do Hibernate
- Mapeia seus comandos para SQL nativos, ou seja, os dialetos do BD em uso no momento
- Vale-se de recursos de cache de primeiro e (opcional) segundo níveis
- Possui sintaxe semelhante ao SQL, para reduzir a curva de aprendizagem

HQL – Características

- Efetua consulta de objetos ou propriedades
- Possui suporte à herança (consultas polimórficas)
- Permite fácil navegação entre associações de objetos
- Além dos tipos de usuário, opera sobre tipos java nativos ou wrappers
  - String, boolean, Date, etc.
  - Coleções Java e de usuário
    - List, Set, Collection, Map, SortedSet, SortedMap
    - Implementando org.hibernate.usertype.UserCollectionType

HQL – Funcionalidades

- Possui inúmeras funções, como no SQL
  - Agregação
    - SUM
    - AVG
    - MIN
    - MAX
    - COUNT
  - Expressões
    - IN, NOT IN, BETWEEN, IS NULL, IS EMPTY, etc.
    - Estruturas CASE (case ... when ... then ... else ... end)
    - Funções de tempo: current_date(), minute(), etc.

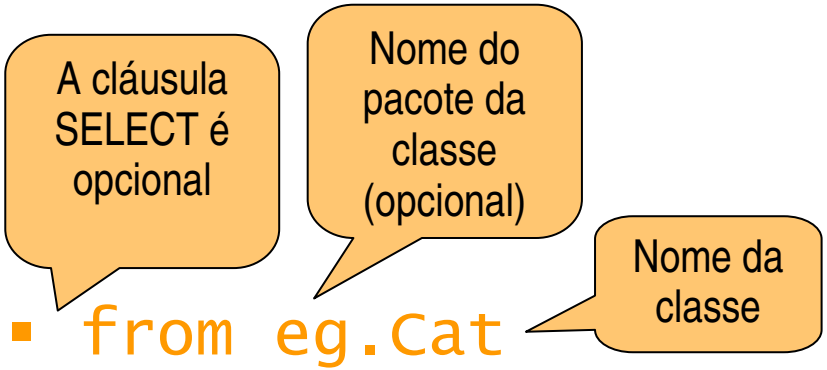
HQL – Funcionalidades

- Mais expressões
  - Qualquer operação definida pelo EJB3-QL
    - substring(), trim(), lower(), upper(), etc.
  - Funções escalares
    - sign(), trunc(), rtrim(), sin(), etc.
  - Funções para coleções
    - size(), minelement(), maxelement(), minindex(), etc.
  - Qualquer variável estática pública do Java
    - Color.RED, com.minhaEmpresa.comum.Sexo.MASCULINO

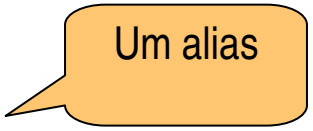
HQL – Sintaxe

- Case insensitive
- Suporte a
  - ORDER BY
  - GROUP
  - HAVING
  - Subqueries (quando o BD suportar)
- Junções implícitas pelas associações
  - Cláusula de junção inferida pelas propriedades identificadoras (id)

HQL – Buscas simples



- `from eg.Cat`
  - Retorna um objeto do tipo *Cat*



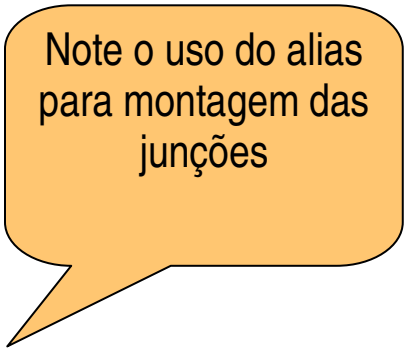
- `from Cat`
- `from Cat as cat`
- `from Cat cat`
  - Formas equivalentes da primeira consulta

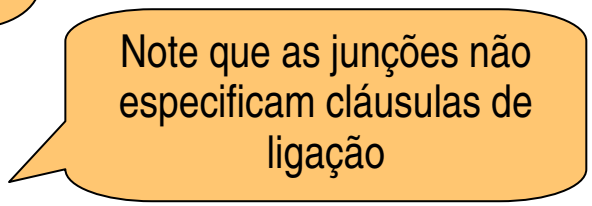
HQL – Mais buscas simples

- `from Formula, Parameter [,…]`
  - Cria um produto cartesiano entre as classes. Cada linha possui uma coleção com dois objetos (um Formula e outro Parameter)

- `from Formula as form, Parameter as param`
  - Mesma coisa, agora com *aliases*

HQL – Junções






```
from Pessoa p
 inner join p.cidade c
 left outer join c.uf
```

- Cria uma junção interna entre uma pessoa e a cidade que ela reside e o resultado é utilizado em uma junção esquerda com a unidade federativa da cidade

HQL – Mais junções


```
from Pessoa p
 inner join p.cidade c
 left outer join c.uf
where c.nome = "Porto Alegre"
```

Note o uso da claúsula WHERE sobre uma propriedade de objeto

- Retorna as pessoas que moram na cidade Porto Alegre, quando c.uf não for igual a null

HQL – Junções externas

Note o produto cartesiano para retornar todas as cidades

```
from Pessoa p
 inner join p.cidade c
 full join c.uf
where c.nome = "Porto Alegre"
```

- Retorna as pessoas que moram na cidade Porto Alegre, mesmo que c.uf seja igual a null

HQL – Exemplos simples

Note que o case é insensitive

Tanto propriedades simples como objetos podem ser retornados

Note o uso de variáveis públicas estáticas (constantes) da API do Java

```
selECT c.name, c.age, c.mate
From Cat as c
WheRE c.mate.name = "Missy"
  AND c.color = Color.WHITE
```

- Retorna o nome e a idade do gato, bem como sua companheira quando esta possuir o nome "Missy" e o pêlo do gato for branco.

HQL – Representação em memória


```
                0          1          2
- selECT c.name, c.age, c.mate
  From Cat as c
  WheRE c.mate.name = "Missy"
    AND c.color = Color.WHITE
```

A mesma consulta de antes

O resultado em memória

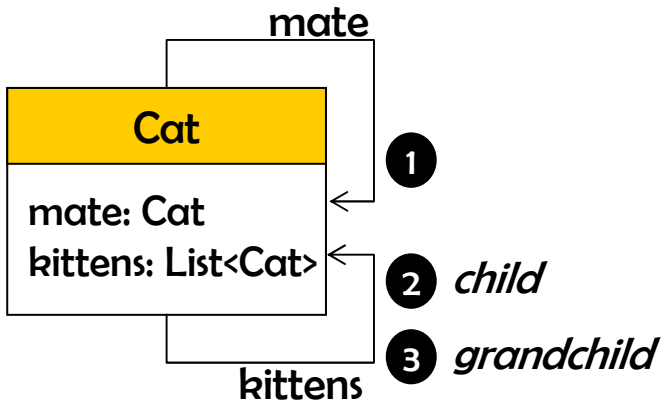
	c.name (String)	c.age (int)	c.mate (Cat)
0	Frajola	2	Missy
1	Belo	3	Missy
n	...	...	...
	0	1	2

Java.util.List

Java.util.List

Propriedades e objetos podem residir simultaneamente no resultado

HQL – Forçando o carregamento



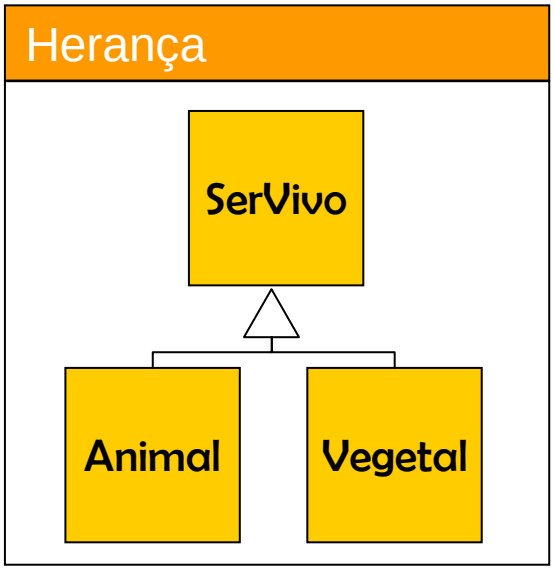
Palavra reservada para inicialização de coleções

```
from Cat as cat
  1inner join fetch cat.mate
  2left  join fetch cat.kittens child
  3left  join fetch child.kittens grandchild
```

- Retorna um gato, inicializando as coleções de seus filhos e netos

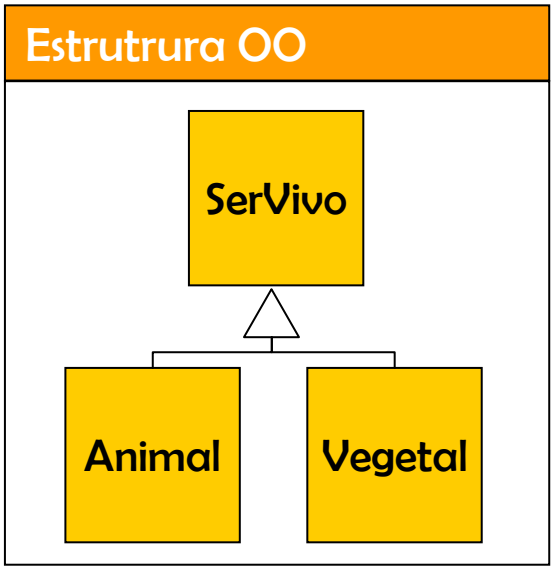
HQL – Herança

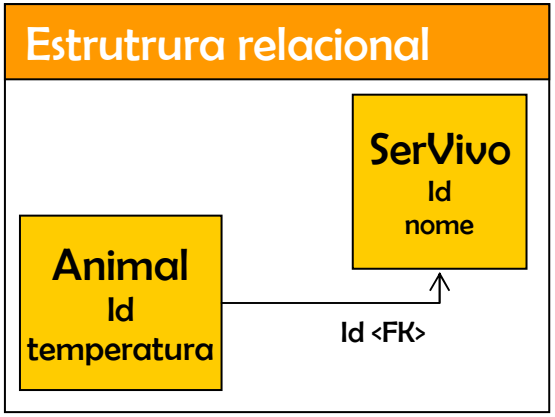
- Query polimórfica (herança)
  - `from SerVivo`
    - Retorna todos seres vivos, animais e vejetais
  - `from Animal`
    - Retorna somente seres vivos do tipo Animal
  - `from Vegetal as v`
    - Retorna somente seres vivos do tipo Vegetal



HQL – Herança e SQL

- O SQL gerado depende da implementação usada na herança, podendo ser:
  - HQL:
  - `from Animal`
  - SQL:
  -
```
select sv.id, sv.nome, a.temp
  from serVivo sv inner join
  animal a on sv.id = a.id
```





Um possível mapeamento, *table per subclass*

HQL – Coleções


- ```
  SELECT p.nome, p.fones
  FROM Pessoa p join p.fones f
  WHERE f.codArea = 51
    AND size(p.fones) > 3
  ```
  - Seleciona o nome e os telefones da pessoa quando o código de área do telefone for 51 e a pessoa possuir mais de 3 telefones

- ```
  from Cat cat
  where exists elements(cat.kittens)
  ```

- ```
  from Player p
  where 3 > all elements(p.scores)
  ```

Usando funções sobre coleções
  ```

HQL e SQL

- HQL:

```
select cust
from Product prod, Store store
  inner join store.customers cust
where prod.name = 'widget'
  and store.location.name in ( 'Melbourne', 'Sydney' )
  and prod = all elements(cust.currentOrder.lineItems)
```

- SQL:

```
SELECT cust.name, cust.address, cust.phone, cust.id,
  cust.current_order
FROM customers cust, stores store, locations loc, store_customers
  sc, product prod
WHERE prod.name = 'widget'
  AND store.loc_id = loc.id
  AND loc.name IN ( 'Melbourne', 'Sydney' )
  AND sc.store_id = store.id
  AND sc.cust_id = cust.id
  AND prod.id = ALL(
    SELECT item.prod_id
    FROM line_items item, orders o
    WHERE item.order_id = o.id
      AND cust.current_order = o.id
    )
```

Comandos diferentes, mesmo resultado

Manipulação de dados

- A partir da versão 3, o Hibernate suporta também a manipulação de dados (INSERT, UPDATE e DELETE)

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlUpdate =
  "update Customer c set c.name = :newName where c.name = :oldName";

int updatedEntities = s.createQuery( hqlUpdate )
  .setString( "newName", newName )
  .setString( "oldName", oldName )
  .executeUpdate();

tx.commit();
session.close();
```

Consultas nativas

- O Hibernate possui suporte à execução de consultas nativas com mapeamento automático de objetos
- `session.createSQLQuery("SELECT ID, NAME, BIRTHDATE, DOG_ID FROM CATS").addEntity(Cat.class).list();`

Retorna uma lista de objetos Cat

- Carregamento customizado

1

```
<sql-query name="person">
    <return alias="pers" class="Person" lock-mode="upgrade"/>
    SELECT NAME AS {pers.name}, ID AS {pers.id}
    FROM PERSON
    WHERE ID=?
    FOR UPDATE
</sql-query>
```

SQL|HQL customizado

2

```
<class name="Person">
    <id name="id">
        <generator class="increment"/>
    </id>
    <property name="name" not-null="true"/>
    <loader query-ref="person"/>
</class>
```

Arquivo de mapeamento usando SQL|HQL customizado

3

```
from Person p
```

Consultas por critério

- Não obstante à HQL, é possível recuperar objetos através de critérios de busca

Definindo critérios para busca de objetos

Um critério de busca simples

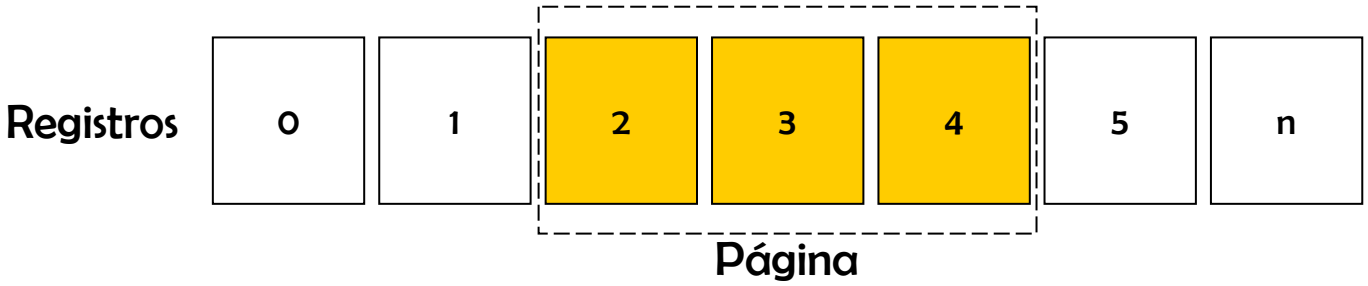
```
Criteria crit = sess.createCriteria(Cat.class);
crit.setMaxResults(50);
List cats = crit.list();
```

Um critério de busca um pouco mais complexo

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .add( Restrictions.or(
            Restrictions.eq( "age", new Integer(0) ),
            Restrictions.isNull("age")
        )
        )
    .list();
```

Paginação

- Consultas podem ser paginadas, como abaixo:

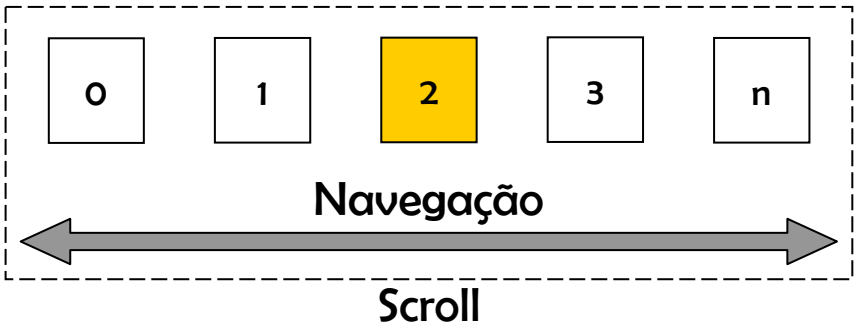


Obtendo uma lista do 20º ao 30º gatos domésticos

```
Query q = sess.createQuery("from DomesticCat cat");
q.setFirstResult(20);
q.setMaxResults(10);
List cats = q.list();
```

Rolagem

- Desde que o driver JDBC suporte, é possível nagevar nos resultados:



Criando um scroll de registros

```
Query q = sess.createQuery("select cat.name,
  cat from DomesticCat cat order by cat.name");
ScrollableResults cats = q.scroll();
//código para navegar no scroll cats
cats.close();
```

Mais funcionalidades

- Para ambientes n camadas, DTOs podem ser gerados automaticamente

Retornando e imprimindo DTOs

```
Query query = session
  .createSQLQuery("SELECT NAME, BIRTHDATE FROM CATS")
  .setResultTransformer(Transformers.aliasToBean(CatDTO.class));
List dtos = query.list();
session.close();
for (CatDTO c : dtos) {
  print(c.getName());
  print(c.getBirthDate());
}
```

Complementando o Hibernate

- O pacote do Hibernate é autocontido e funcional, porém, outros dois pacotes fazem parte do projeto
  - Hibernate Annotations
  - Entity Manager

Hibernate Annotations

- Por padrão, todo o processo de mapeamento do Hibernate é feito através de arquivos XML
- Porém, com o advento da JSR175, anotações Java podem ser usadas para inserir as informações de mapeamento
- Vantagens
  - Menor número de arquivos
  - Pode ser compilada
- Desvantagens
  - Necessita recompilação em caso de alterações
    - O que é minimizado pelas modernas IDEs e o uso de ferramentas como o ANT

Hibernate Annotations

- Características
  - Release estável
  - Integração com a JSR220 (Java Persistence API)
  - Agrega extensões úteis
    - Cache
    - Cascateamento
    - outras

Hibernate Annotations

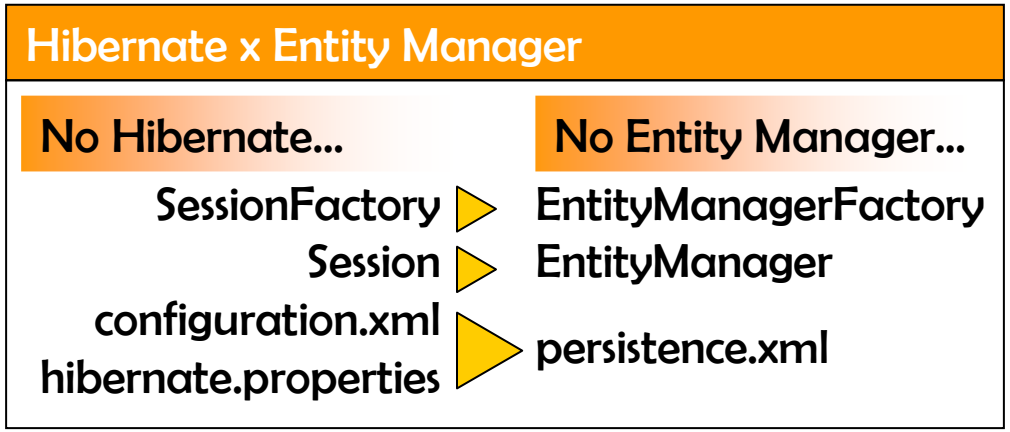
- Validator
  - Conjunto de anotações que visam:
    - Verificar constraints sobre objetos em memória
    - Incorporar constraints no esquema de dados do BD
  - Exemplos
    - @Length(min,max)
    - @Max(value)
    - @Min(value)
    - @NotNull
    - @Pattern(regex)
    - @Range(min,max)
    - @Email
    - @MinhaConstraint

Hibernate Annotations

- Validators, funcionamento
  - Adiciona-se as anotações de validação sobre as propriedades dos objetos
  - Via aplicação cliente
    - Encapsula-se a classe anotada em um `ClassValidator`
    - Invoca-se `getInvalidValues` sobre um objeto da classe
    - Exibe-se os resultados, que são i18n
  - Via listeners de sessão (atuam como interceptadores)
    - Configura-se listeners para operações de INSERT ou UPDATE
    - Exceptions em runtime são lançadas, caso haja violação nas constraints

Entity Manager

- Extensão do Hibernate que provê as funcionalidades descritas no padrão JSR220 (EJB3)
  - Na prática, é usado para permitir uma fácil migração do Hibernate para o EJB3



Ferramentas

- O framework Hibernate representa o kernel de um sistema de mapeamento OO-ER
- Possui, ainda, três ferramentas básicas de trabalho
- Outras ferramentas também estão disponíveis para facilitar seu uso

Geradores integrados

- SchemaExport
  - Geração da base de dados a partir das classes mapeadas (anotações ou XML)
- SchemaUpdate
  - Atualização incremental da base de dados a partir das classes mapeadas (anotações ou XML)
- SchemaValidator
  - Validação das estruturas OO frente ao esquema do banco de dados

Integração com o ANT

- O Apache ANT é uma consagrada ferramenta de construção para Java
- O Hibernate possui extensões que permitem sua execução via scripts do ANT

Gerando o script do banco de dados

```
<target name="bd.ddl" description="Gera a DDL do BD a partir o arquivo de persistencia." depends="per
    <hibernatetool destdir="${build.dir}">

        <ejb3configuration />

        <classpath>
            <path location="${build.dir}/${application.name}.jar" />
        </classpath>

        <hbm2ddl export="false" outputfilename="${application.name}_sql_create.ddl" />
        <hbm2ddl export="false" update="true" outputfilename="${application.name}_sql_update.ddl" />

    </hibernatetool>
</target>
```

O Hibernate dentro de IDEs

- Plugins diversos permitem utilizar o Hibernate dentro de IDEs, como o Eclipse
- O projeto Hibernate Tools (ex Hibern8IDE) é um exemplo que habilita
  - Operação visual
  - Geração automática do arquivo de configuração
  - Console para execução de consultas HQL
  - Previsão dos comandos SQL produzidos

Hibernate e .NET

- Desde o ano passado, o Hibernate também está disponível para desenvolvedores .NET, é o NHibernate

Dicas para desenvolvimento

- Utilize abordagem TopDown
  - Pense e modele OO, persista e recupere objetos, não colunas
- Automatize com ANT
  - Bons scripts valem mais que várias ferramentas
- Use plugins visuais, mas primeiro entenda o conceito
- Tenha uma IDE configurada
  - Templates de código são fundamentais
- Siga as recomendações
  - Pool, gerenciamento de sessões, ajuste fino na configuração, etc.

Dicas para desenvolvimento

- Não reinvente a roda
  - Seja assíduo nos fóruns, leia artigos, estude a API, depure os exemplos, leia os livros
- Pense OO, mas não esqueça o ER
  - Afinal, os objetos ainda são persistidos em bases relacionais
- Entenda um pouco mais de BD
  - Indices, sintaxes nativas, normalização, etc. são cruciais para melhorar a performance

Dicas para desenvolvimento

- Anotações podem ser compiladas, XML não
- Aprenda HQL, mas não esqueça SQL
- Verifique o log
- Analise as estatísticas e afine o sistema
- Use cache sempre que possível
- Em ambientes gerenciados, delegue as transações

Comentários finais

- O Hibernate não carrega dados desnecessários
  - Em outras palavras é a sua configuração está errada!
- Usar um framework de mapeamento OO-ER não implica em ter um BD desorganizado (sem PKs, FKs, etc.)
- Operações "pesadas" (ex.: DELETE n-n) podem ser otimizadas, mas com cuidado
- Usar um framework de mapeamento OO-ER não implica em "vou esquecer que tenho um banco de dados!"
- Lei do 80/20 (eu diria até 95/5)
  - Nem sempre a HQL e a busca por critérios será suficiente, daí usamos SQL e mapeamento automático

Espaço comum

- Participe!

Referências

- **Java Persistence with Hibernate.** Bauer, Christian; King, Gavin. Manning, 2007.
- **Hibernate in Action.** Bauer, Christian; King, Gavin. Manning, 2005.
- **Hibernate: A J2EE™ Developer's Guide.** Iverson, Will. Addison Wesley, 2004.
- **Hibernate Quickly.** Peak, Patrick; Heudecker, Nick. Manning, 2006.
- **Hibernate Relational Persistence for Idiomatic Java API.** Versão 3.2.1.

Fim

Hibernate

Uma visão geral sobre o framework padrão *de fato* para mapeamento objeto-relacional

Slides Auxiliares

- A partir daqui, somentes slides auxiliares (figuras, textos, rascunhos, etc.)