Mapeamento O/R e Hibernate

Luiz Fernando Rodrigues
lfrodrigues@teccomm.les.inf.puc-rio.br

Agenda – Mapeamento O/R

- Conceitos Básicos
  - Shadow Information
- Mapeamento de relações de objetos
  - Tipos de Relacionamento
  - Formas de Mapeamento
  - Considerações
    - Coleções ordenadas
    - Relacionamento Recursivo
    - Escopo de Propriedades
    - Lazy
- Mapeamento de estruturas de herança
  - Abordagens Possíveis

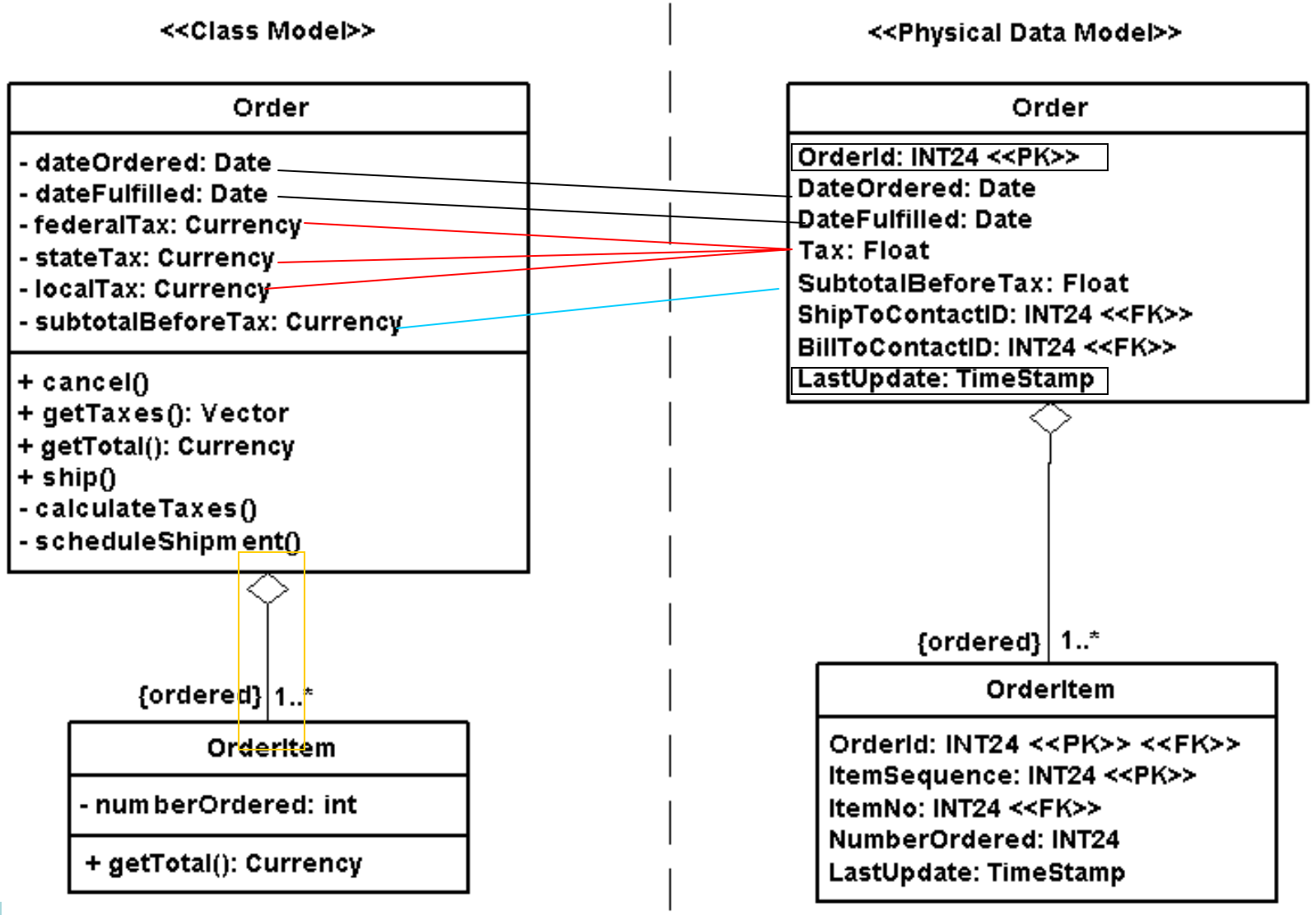
Agenda - Hibernate

- Hibernate
  - Conceitos
- Arquitetura
  - Elementos
- Classes Persistentes
  - POJOs
- Mapeamentos
  - Básico
  - Coleções
  - Associações
  - Herança
- Trabalho com Objetos
- Sessões e Transações

Mapeamento O/R – Conceitos

- Mapeamento mais simples
  - Atributo -> coluna
  - Isso infelizmente raramente acontece
- Outras considerações
  - Associações
  - Herança
  - Ordenação
  - Recursão
  - Atributos não mapeados
  - Shadow Information

Exemplo



Shadow Information

- Informação mantida no objeto que vai além de seu domínio
- São necessárias para persistência
- Ex
  - LastUpdate
  - OrderId

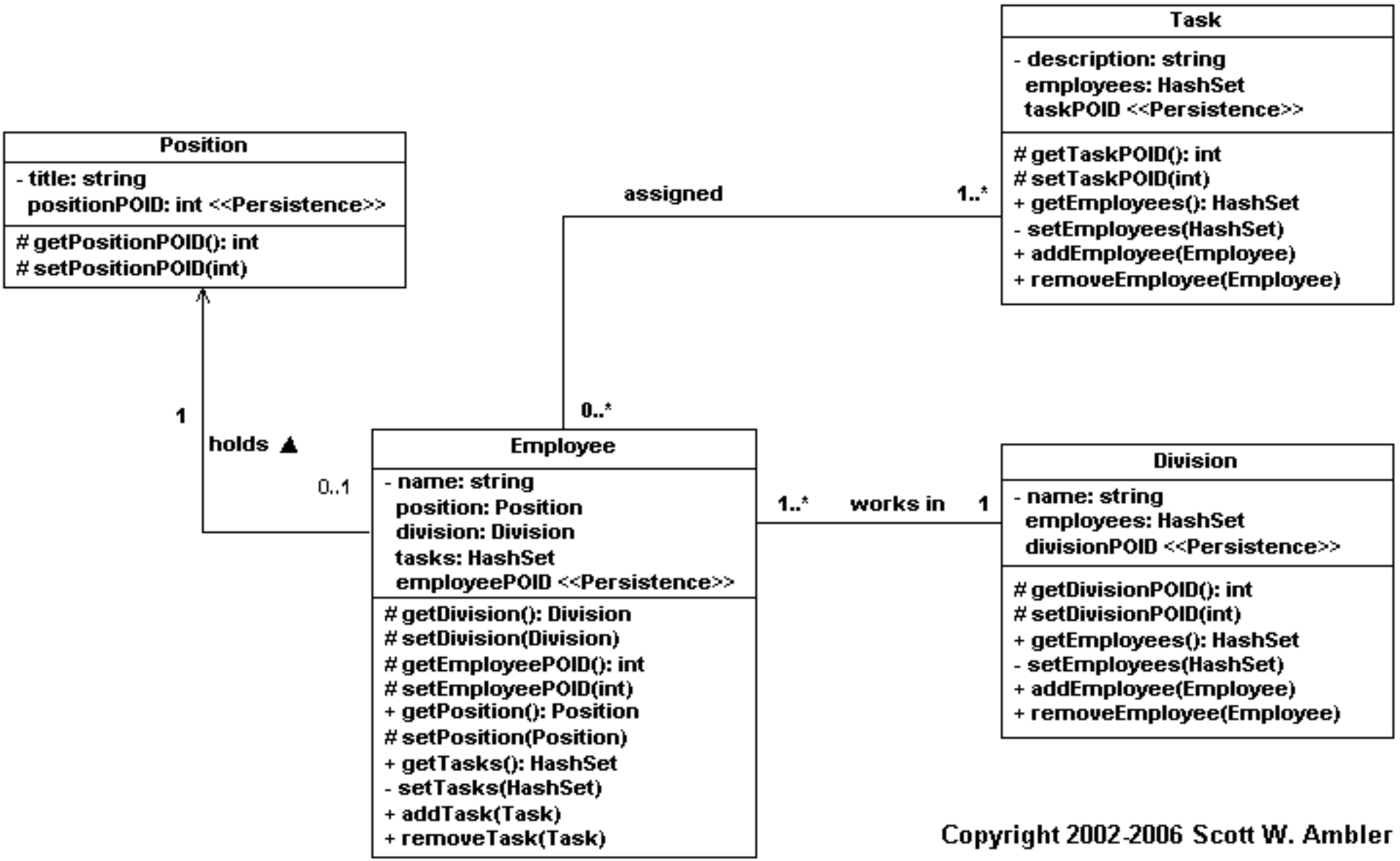
Relacionamentos

- Associações, Agregações e Composições
- Multiplicidade
  - One-to-One
  - One-to-Many
  - Many-to-Many
- Navegabilidade
  - Uni-direcional
  - Bi-direcional
- 6 combinações possíveis
- E/R todas as associações são bi-direcionais

Implementação em Objetos

- One-to-One
  - Referencia para objeto relacionado
- One-to-Many , Many-to-Many
  - Atributo de Coleção
- Uni-direcional
  - Apenas um objeto conhece o outro
- Bi-direcional
  - Ambos objetos se conhecem

Exemplo Objetos



Implementação Relacional

- Relacionamentos
  - Uso de chaves estrangeiras
- One-to-one
  - Uma das tabelas guarda a chave
- One-to-Many
  - Tabela de “One” referencia tabela de “Many”
- Many-to-Many
  - Tabela de Associação

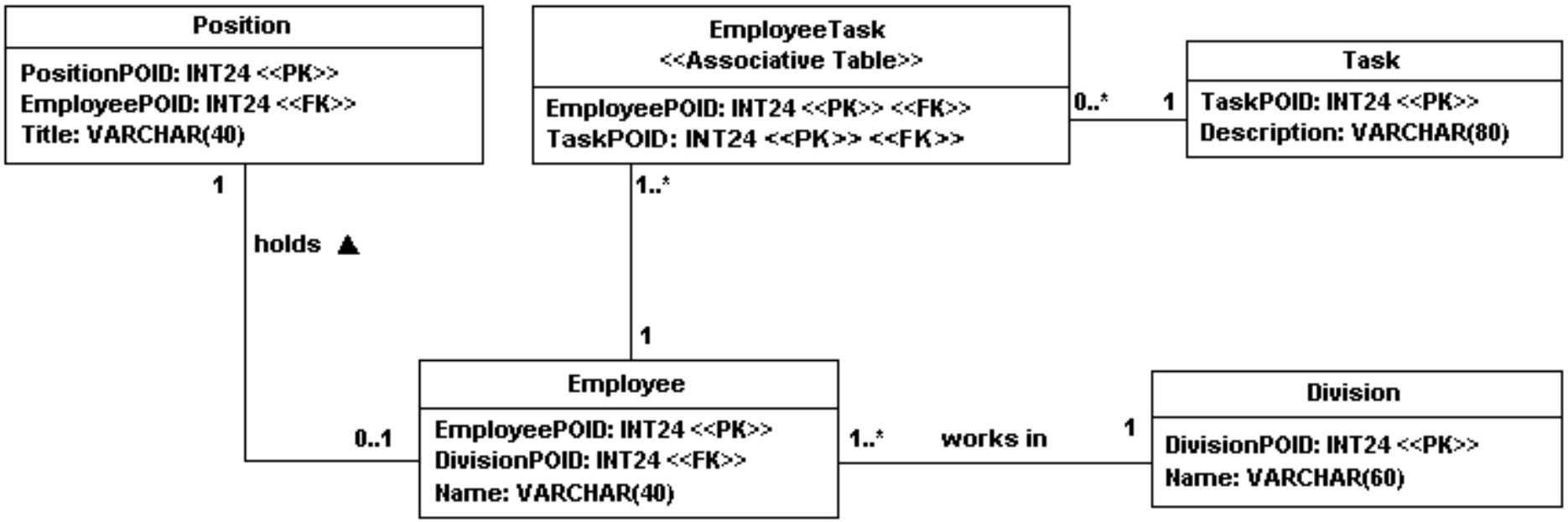
Implementação Relacional

- Sempre Bi-Direcional

SELECT * FROM Position, Employee WHERE
Position.EmployeePOID Employee.EmployeePOIDHad

SELECT * FROM Position, EmployeeWHERE
Position.PositionPOID = Employee.PositionPOID

Exemplo Relacional



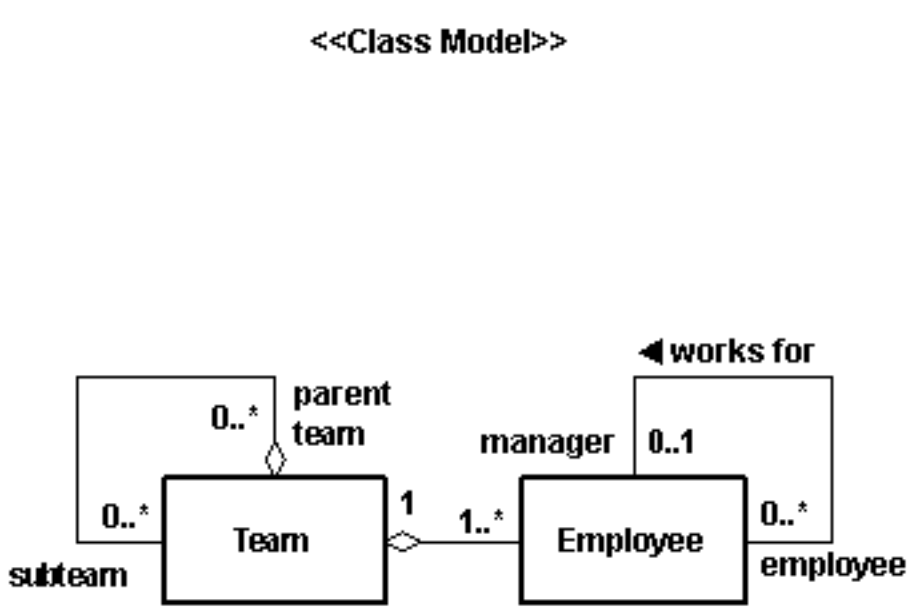
Mapeamento de Coleções Ordenadas

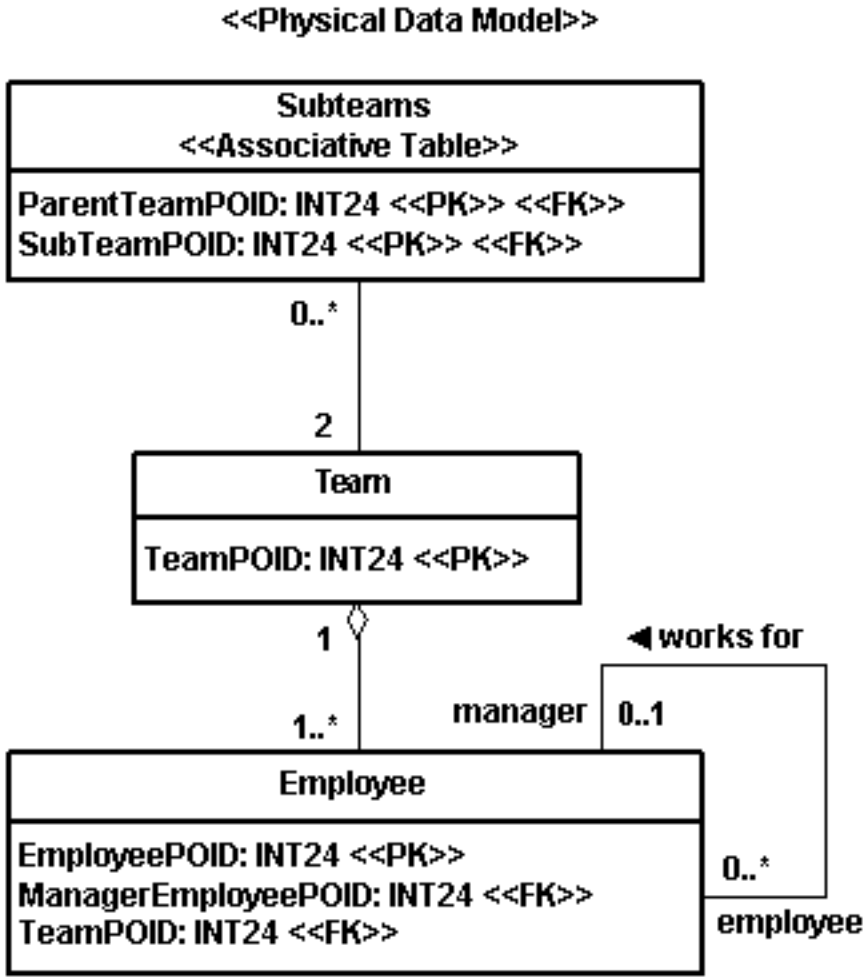
- Considerações
  - Dados devem ser lidos em seqüência
  - Não é recomendado que a seqüência seja uma chave
    - Ao incluir um novo elemento entre dois elementos é necessário nesse caso trocar as outras chaves
  - Gaps de seqüência podem ser maior que 1
    - Quanto maior o gap mais tardio sera a refatoração das tabelas

Mapeamento de Relacionamentos Recursivos

- Igual ao mapeamento não recursivo

Exemplo

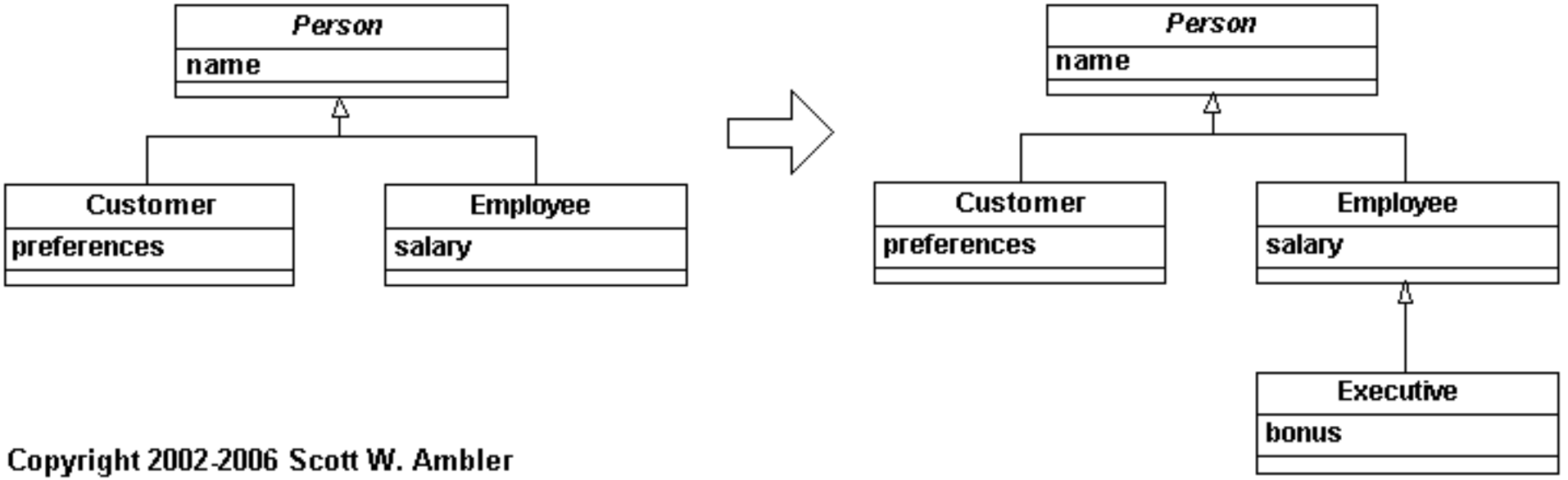




Mapeamento de estruturas de herança

- Abordagens
  - Toda hierarquia em uma única tabela
  - Cada classe concreta com sua própria tabela
  - Cada classe com sua própria tabela
  - Estrutura genérica de mapeamento

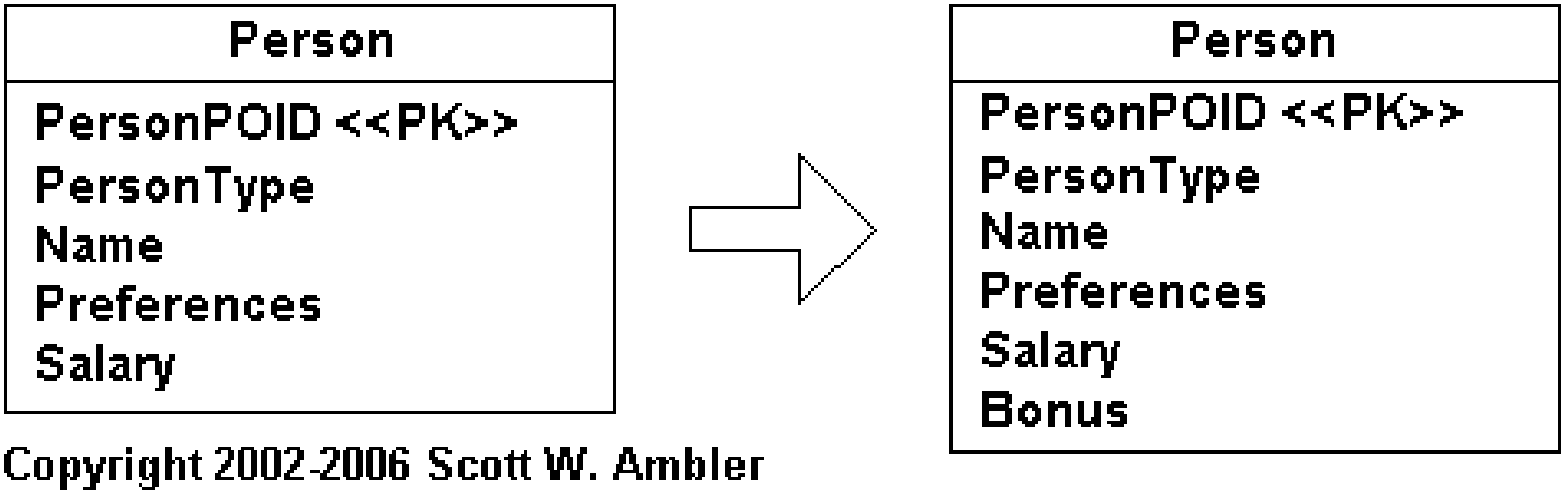
Exemplo



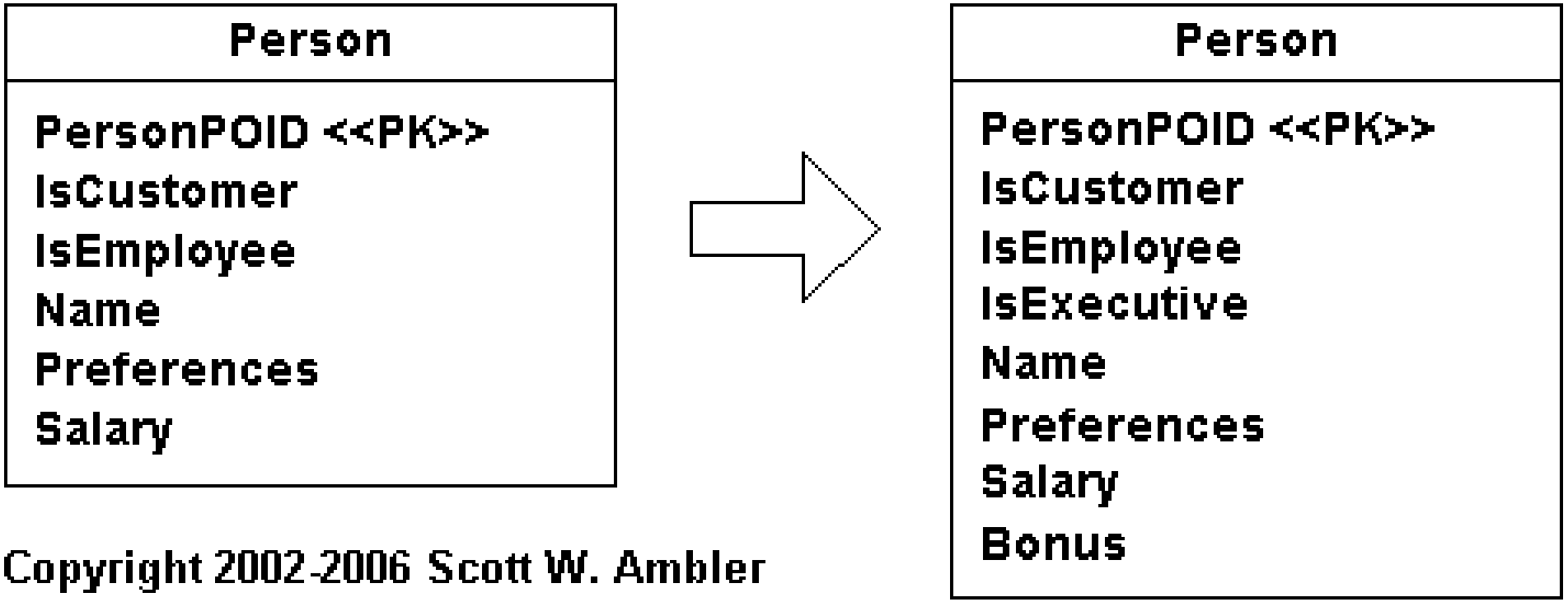
Toda hierarquia em uma única tabela

- Todos os atributos são armazenados em uma única tabela
- Necessita um identificador de tipo
  - Um campo tipo
  - Vários campos

Exemplo – Um campo tipo



Exemplo – Vários campos



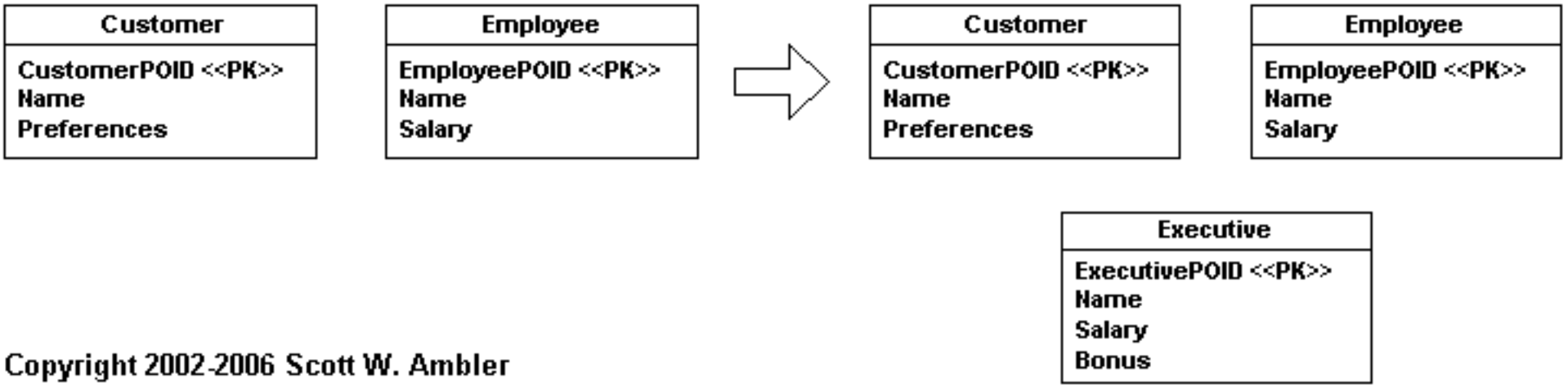
Análise da Abordagem

- Vantagem
  - Simples implementação
  - Facil inclusao de classes
  - Suporte de polimorfismo
  - Acesso rapido
- Desvantagens
  - A modificação de uma classe pode afetar outras classes na hierarquia
- Uso
  - Pequeno overlap entre os tipos da hierarquia

Cada classe concreta com sua própria tabela

- Uma tabela por classe do sistema

Exemplo



Análise da Abordagem

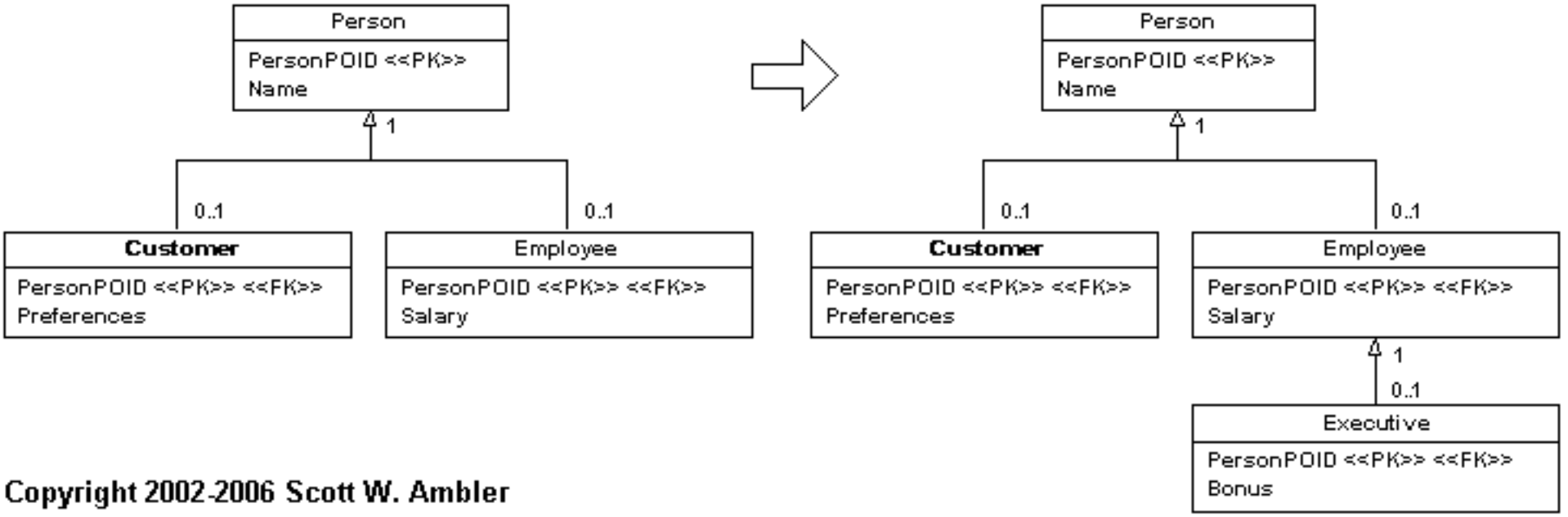
- Vantagem
  - Boa performance para acesso
- Desvantagem
  - Modificação de uma classe acarreta na modificação de sua tabela e todas as classes filhas
  - Dificuldade de manter diversos papeis para um único objeto
  - Dificuldade de modificar um papel de um objeto
- Uso
  - Quando a troca de tipos e/ou overlap de tipos é rara

Cada classe com sua própria tabela

- Cada classe possui sua própria tabela

- Informação de um objeto pode ficar em diversas tabelas

Exemplo



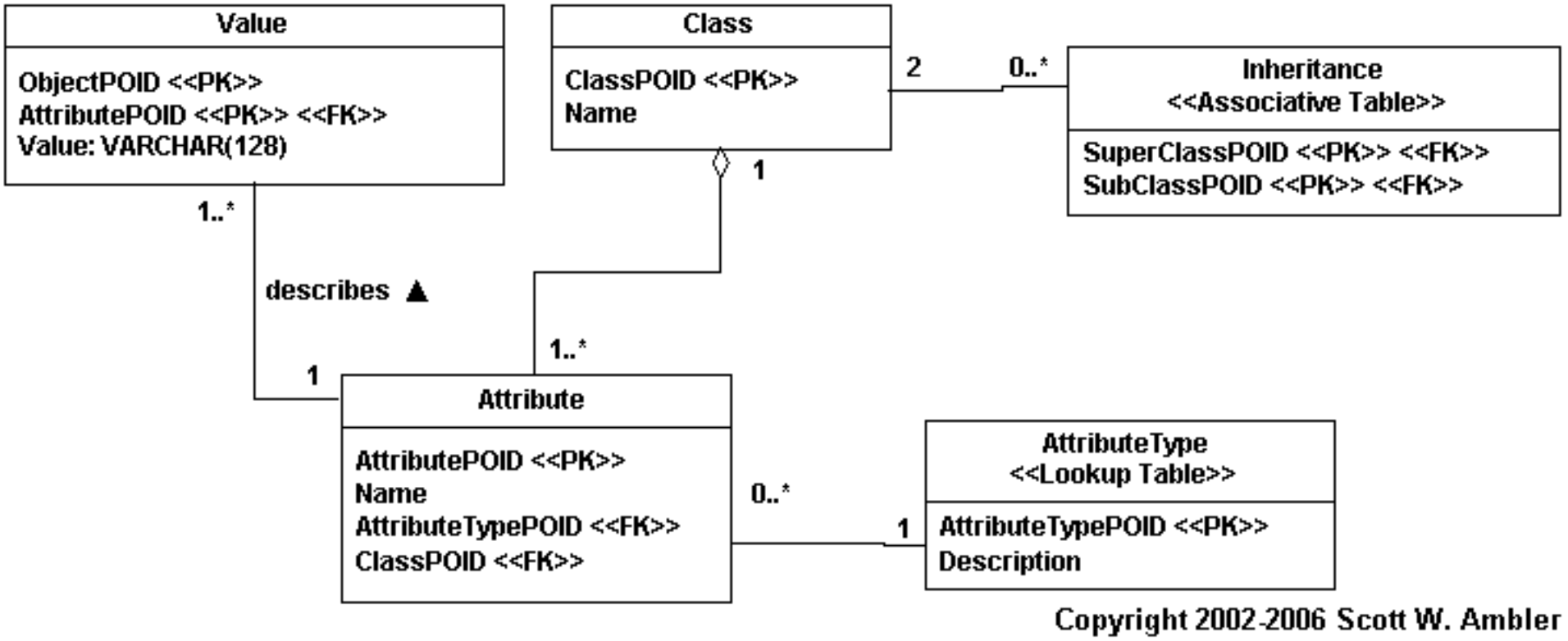
Análise da Abordagem

- Vantagem
  - Facil compreensão
  - Bom suporte para polimorfismo
  - Facilidade para modificar superclasses
  - Crescimento de dados é proporcional ao número de objetos
- Desvantagem
  - Muitas tabelas
  - Potencialmente lento
- Uso
  - Quando o overlap entre os tipos é significativo
  - Mudanca nos tipos é comum

Estrutura Genérica

- Não específica para herança
  - Aplicavel para qualquer forma de mapeamento

Estrutura


Value
ObjectPOID <<PK>>
AttributePOID <<PK>> <<FK>>
Value: VARCHAR(128)
Class
ClassPOID <<PK>>
Name
2
0..*
Inheritance
<<Associative Table>>
SuperClassPOID <<PK>> <<FK>>
SubClassPOID <<PK>> <<FK>>
1
1..*
describes
1..*
1
Attribute
AttributePOID <<PK>>
Name
AttributeTypePOID <<FK>>
ClassPOID <<FK>>
0..*
1
AttributeType
<<Lookup Table>>
AttributeTypePOID <<PK>>
Description
Copyright 2002-2006 Scott W. Ambler

Análise da Abordagem

- Vantagem
  - Facil inclusao de novos tipos
- Desvantegem
  - Dificuldade de implementação
  - Funciona apenas com pequenas quantidades de dados
- Uso
  - Aplicações complexas com pequenas quantidades de dados

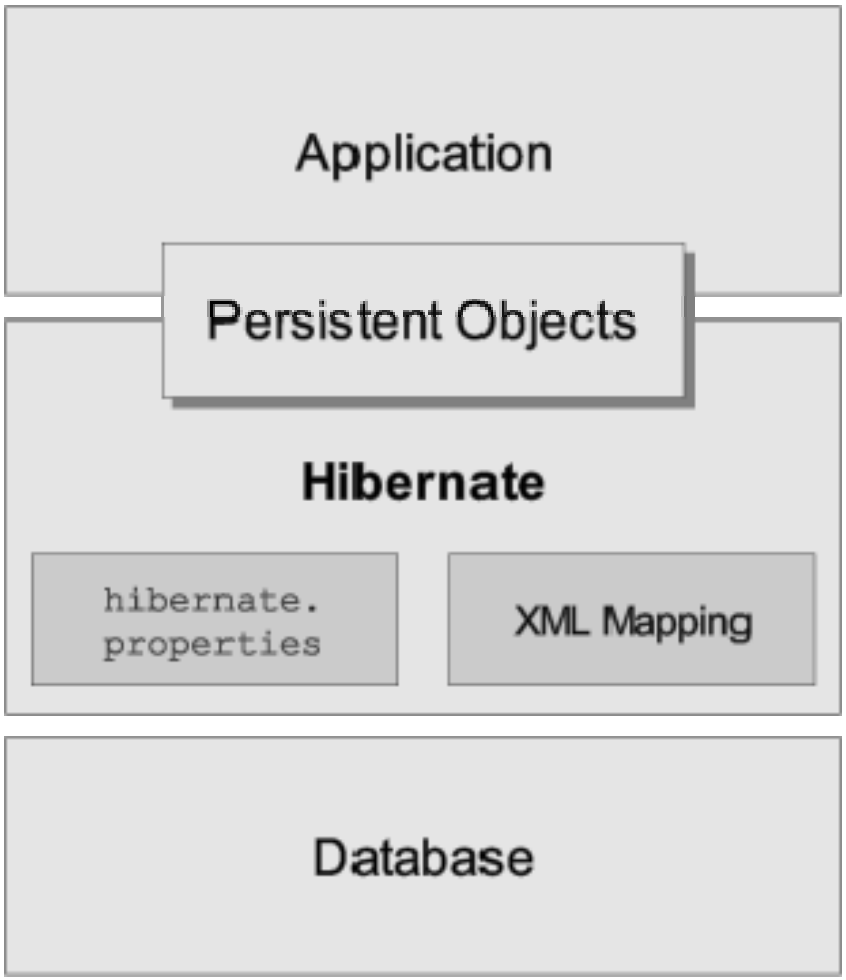
Hibernate

Hibernate – O que é?

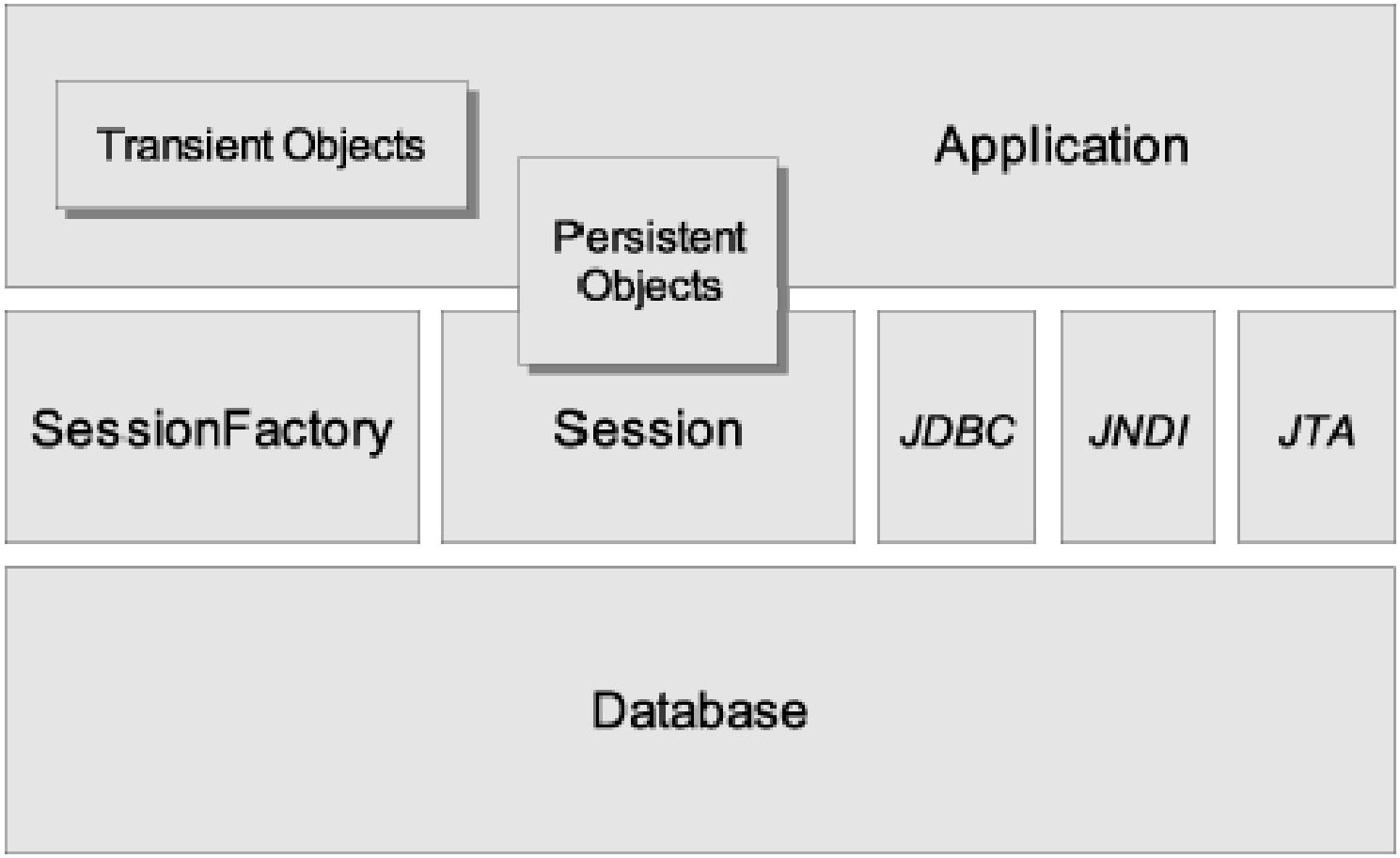
- Ferramenta para mapeamento O/R em ambientes Java
- Busca de dados
  - HQL
  - Criteria Queries
- Facilidade de uso
  - Controle de transações
  - Independência quanto ao tipo de base de dados

Hibernate - Arquitetura

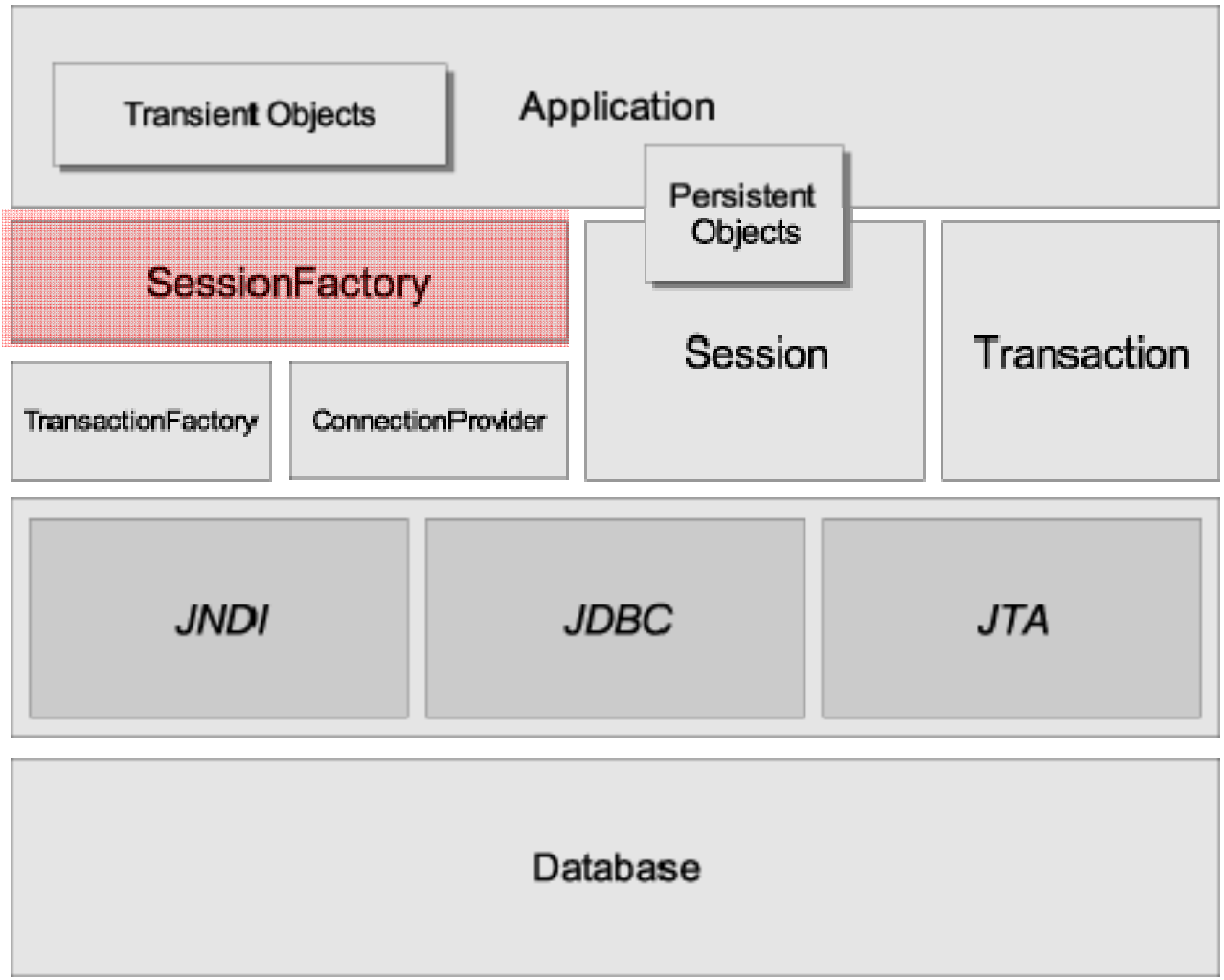
Arquitetura - Overview



Arquitetura – Resumida



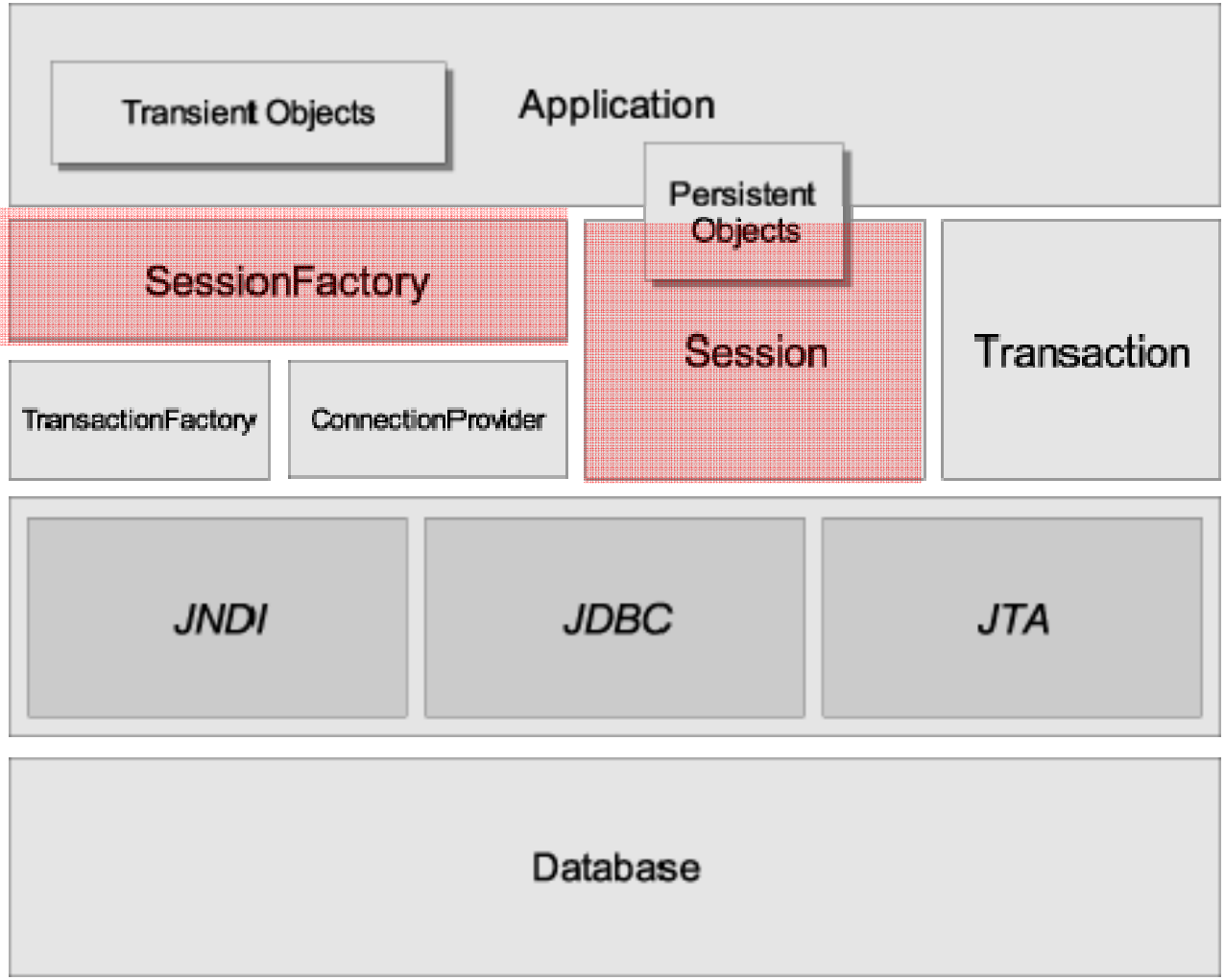
Arquitetura – Detalhada



SessionFactoty

- org.hibernate.SessionFactory
- Threadsafe cache de mapeamentos para uma base de dados
- Fábrica para Session e cliente ConnectionProvider
- Cache para reutilização de transações

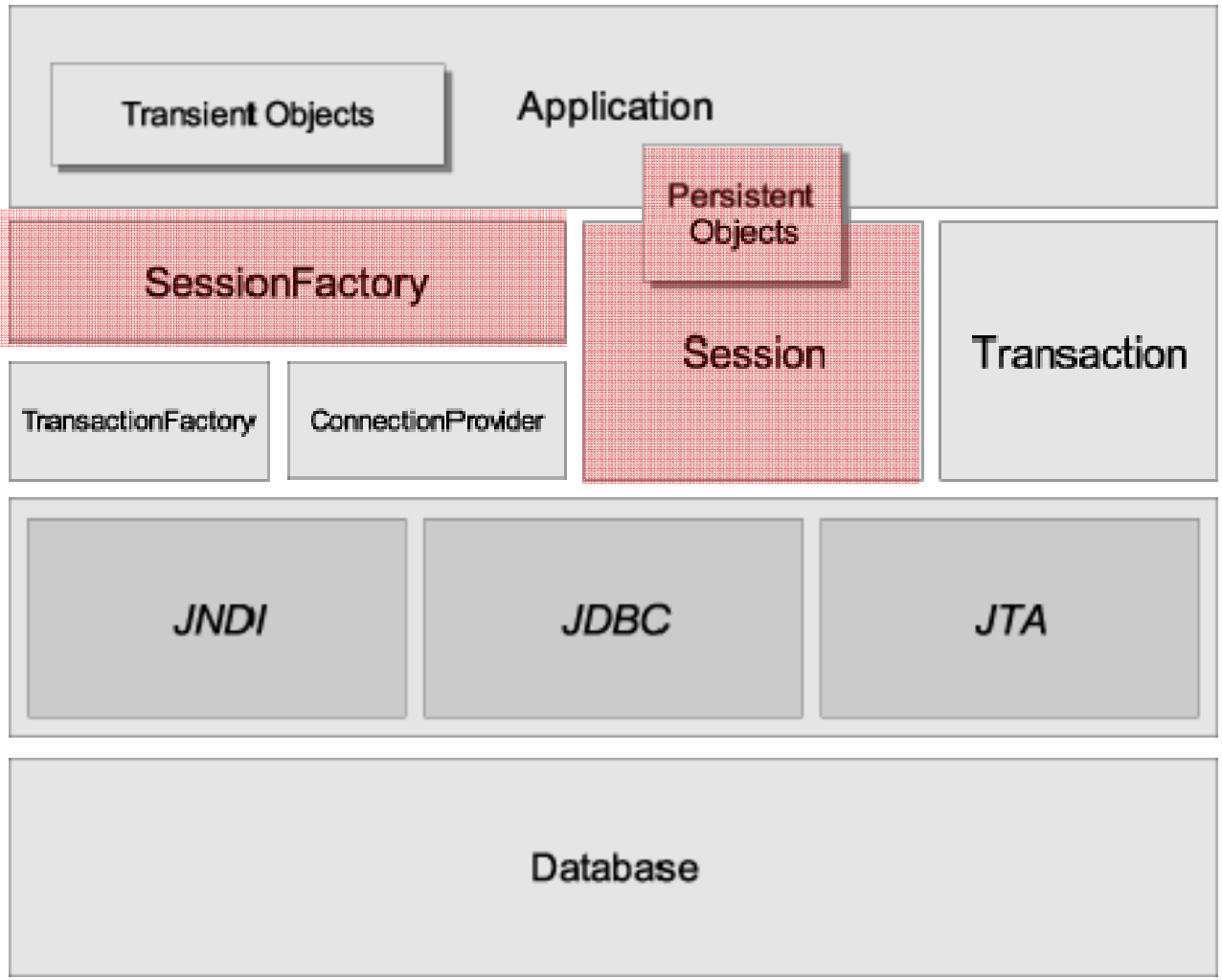
Arquitetura – Session



Session

- org.hibernate.Session
- Objeto de “vida-curta” e thread única
- Representa a troca de informações entre a aplicação e um esquema de armazenamento de dados
- Fábrica de transações
- Wrapper para conexões JDBC
- Mantém referencia para os objetos persistentes

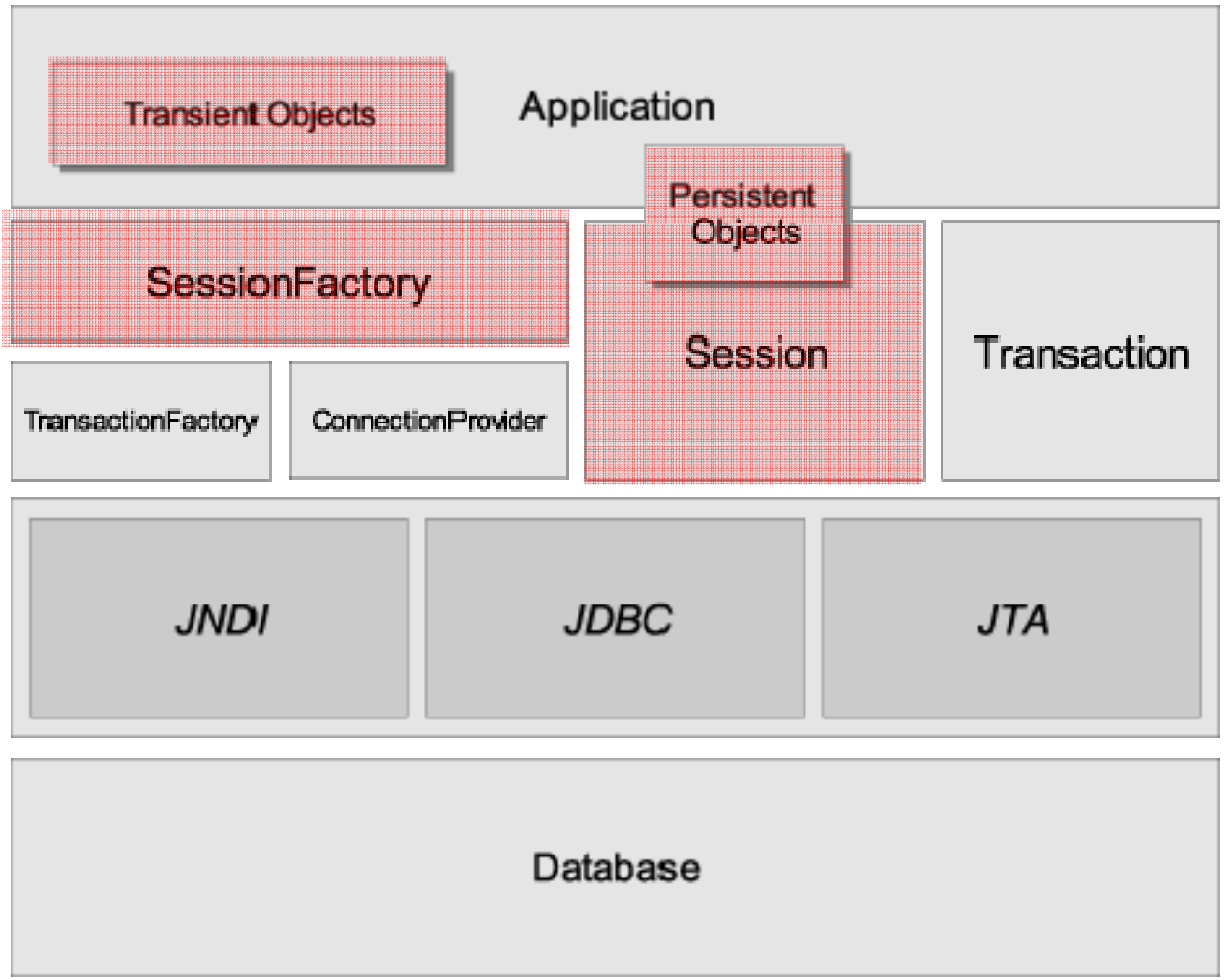
Arquitetura – Persistent Objects



Persistent Objects

- Objetos de “vida curta”
- pertencentes a lógica da aplicação
- JavaBeas/POJO
- Associados a uma sessão

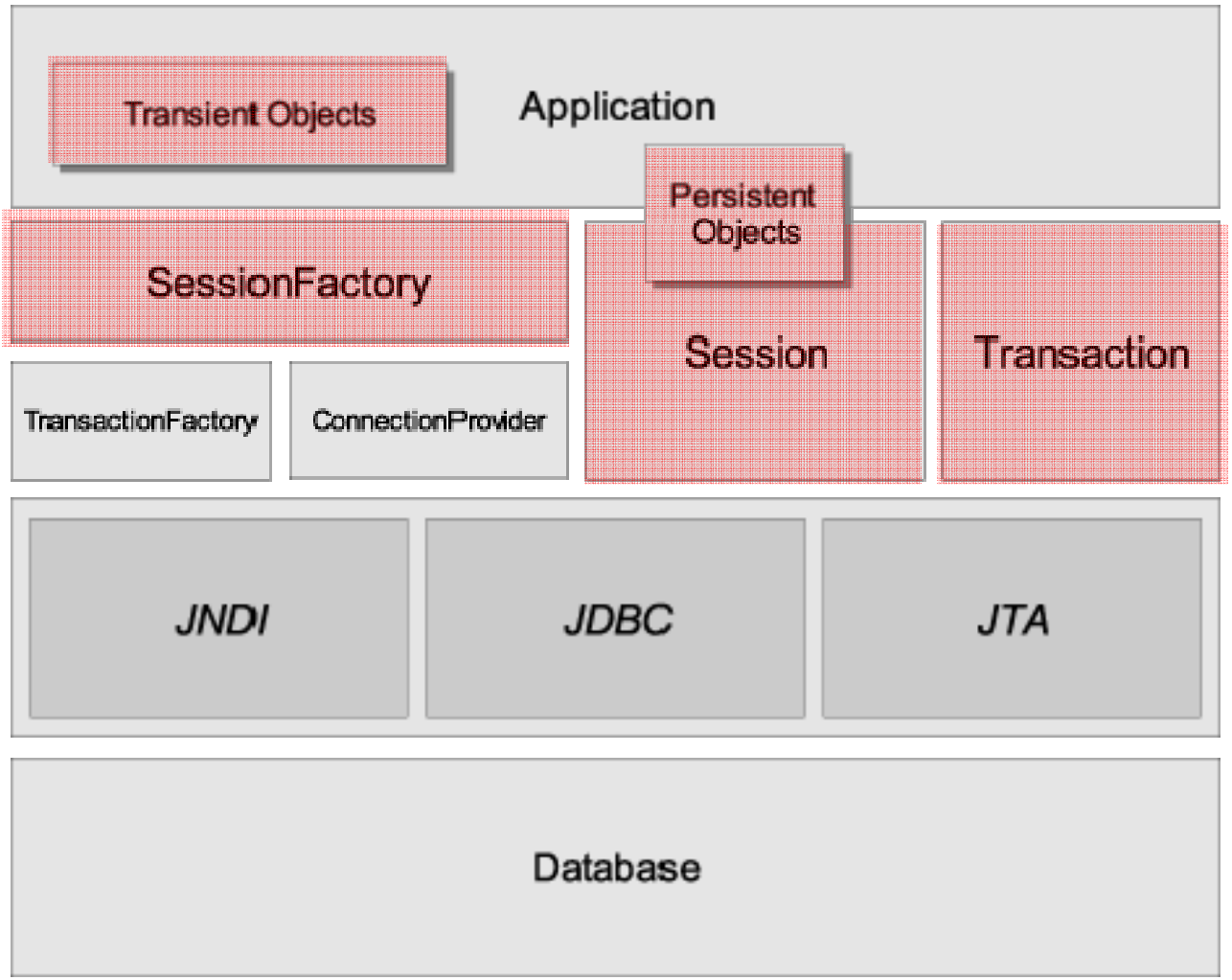
Arquitetura – Transient Objects



Transient e Detached Objects

- Objetos da lógica da aplicação
- Transient
  - Nunca foram associados a sessão alguma.
- Detached
  - Já foram associados a uma sessão
  - Possuem identidade de persistência
    - Dados na base
  - Hibernate não garante a consistência dos objetos java “detacheds” com aqueles persistidos

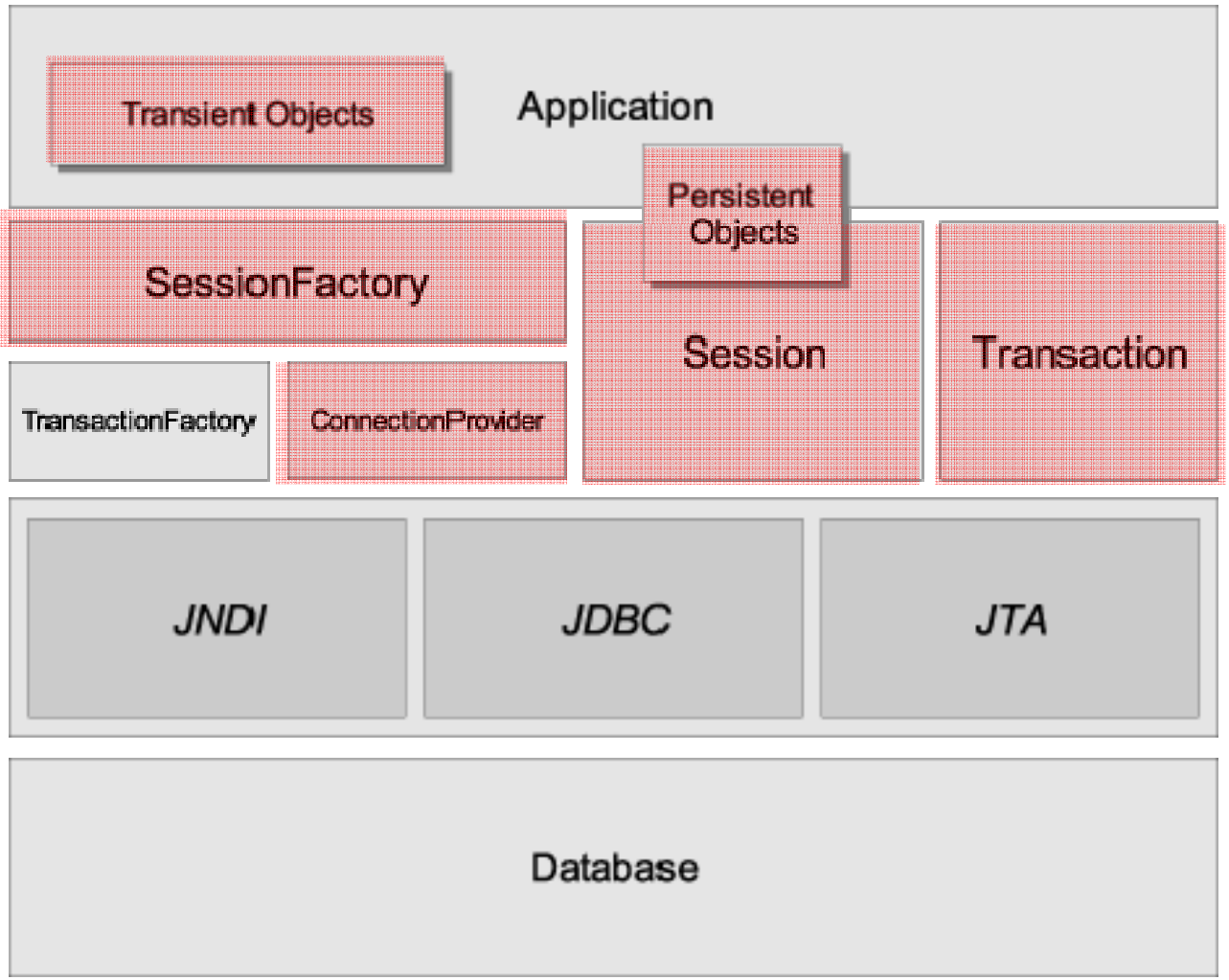
Arquitetura – Transaction



Transaction

- Recurso Opcional
- Objeto de “vida curta”
- Utilidade em unidades específicas de trabalho
- Umas sessão pode utilizar diversas transações

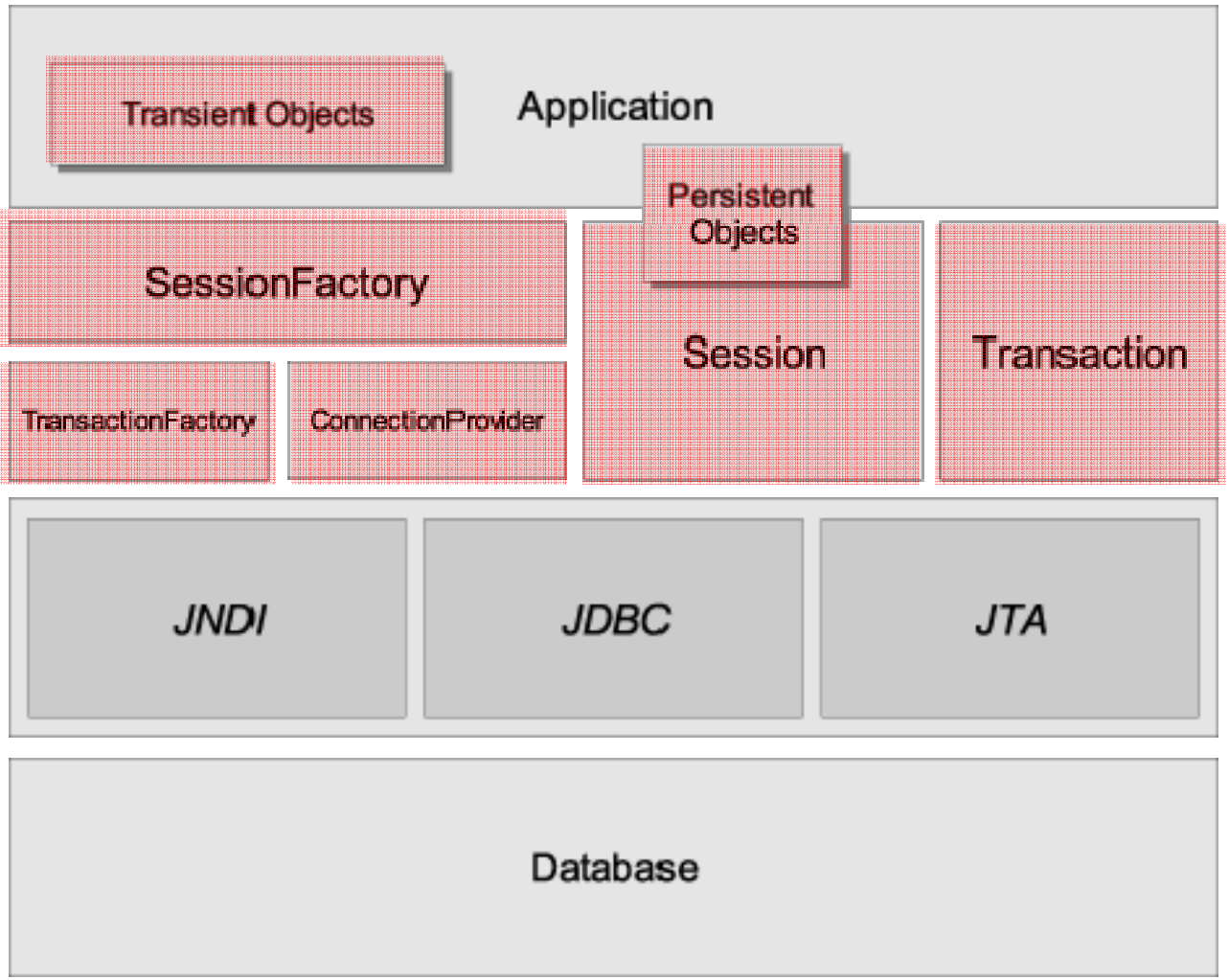
Arquitetura – ConnectionProvider



ConnectionProvider

- org.hibernate.connection.ConnectionProvider
- Recurso Opcional
- Fábrica para conexões JDBC
- Não é exposta para aplicação
- Pode ser estendida/implementada pelo desenvolvedor

Arquitetura – TransactionFactory



TransactionFactory

- org.hibernate.TransactionFactory
- Recurso opcional
- Não exposta a aplicação
- Pode ser estendida/implementada pelo desenvolvedor

Estados das Instancias

- Estados são relativos a um contexto de persistência
  - Sessão
- Transient
  - Nunca associado a um contexto
  - Não possui identificador de persistência (PK)
- Persistent
  - Associada a um contexto
  - Possui um identificador de persistência (PK)
    - Possivelmente possui uma equivalência com a base
  - O Hibernate garante a consistência de objetos persistidos com aqueles em memória
- Detached
  - Já foram associados a uma sessão
  - Possuem identidade de persistência
    - Dados na base
  - Hibernate não garante a consistência dos objetos java “detacheds” com aqueles persistidos

Hibernate – Classes Persistentes

Classes Persistentes

- Implementam entidades da lógica de negócio
- Nem todas as instancias de uma classe persistente estão no estado de persistência
  - Transient
  - Detached
- POJOs
  - Plain Old Java Object

POJO

- Construtor sem argumentos
  - Instanciação por reflexão
  - Geração de Proxy
- Identificador
  - Mapeamento de PK
- Classes não finais
  - Proxies necessitam de classes não finais para poder estende-las
- Get/Sets
  - Evita manipulação direta dos campos

POJO - Exemplo


```
package eg;
import java.util.Set;
import java.util.Date;

public class Cat {
    private Long id; // identifier

    private Date birthdate;
    private Color color;
    private char sex;
    private float weight;
    private int litterId;

    private Cat mother;
    private Set kittens = new HashSet();

    private void setId(Long id) {
        this.id=id;
    }
    public Long getId() {
        return id;
    }

    void setBirthdate(Date date) {
        birthdate = date;
    }
```

```
    public Date getBirthdate() {
        return birthdate;
    }
    void setWeight(float weight) {
        this.weight = weight;
    }
    public float getWeight() {
        return weight;
    }

    public Color getColor() {
        return color;
    }
    void setColor(Color color) {
        this.color = color;
    }

    void setSex(char sex) {
        this.sex=sex;
    }
    public char getSex() {
        return sex;
    }

    void setLitterId(int id) {
        this.litterId = id;
    }
```

```
    public int getLitterId() {
        return litterId;
    }

    void setMother(Cat mother) {
        this.mother = mother;
    }
    public Cat getMother() {
        return mother;
    }
    void setKittens(Set kittens) {
        this.kittens = kittens;
    }
    public Set getKittens() {
        return kittens;
    }

    // addKitten not needed by
    //  Hibernate
    public void addKitten(Cat kitten) {
                kitten.setMother(this);
                kitten.setLitterId(
                    kittens.size() );
                kittens.add(kitten);
    }
}
```

POJO - Herança

- Obedecer as duas primeiras regras
  - Construtor
  - Identificador

```
package eg;

public class DomesticCat extends Cat {
        private String name;

        public String getName() {
                return name;
        }
        protected void setName(String name) {
                this.name=name;
        }
}
```

equals() e hashCode()

- Utilização de Sets
- Realocação e desalocação em sessão
  - reattachment e detached
  - Sessões utilizam conjuntos de objetos

- Respeitar Contrato de Sets
  - Não implementar equals()/haschode() com identificador
  - Utilizar lógica do negócio

Exemplo - equals() e hashCode()


```
public class Cat {
    ...
    public boolean equals(Object other) {
        if (this == other) return true;
        if ( !(other instanceof Cat) ) return false;

        final Cat cat = (Cat) other;

        if ( !cat.getLitterId().equals( getLitterId() ) ) return false;
        if ( !cat.getMother().equals( getMother() ) ) return false;

        return true;
    }

    public int hashCode() {
        int result;
        result = getMother().hashCode();
        result = 29 * result + getLitterId();
        return result;
    }
}
```

Hibernate – Mapeamento

Mapeamento

- Devemos informar ao Hibernate como relacionar o modelo de objetos com o modelo relacional
- Arquivos de configuração
  - Hibernate-mapping
    - Descrevem como são feitos os mapeamentos
  - Hibernate-configuration
    - Descrevem algumas configurações de como o hibernate deve operar
      - Driver
      - Localização da Base de Dados
      - Tipo de Dialeto
      - Tipo de Conexão
      - ...
      - **Arquivos de Mapeamento Disponíveis**

Arquivo de Mapeamento

- Arquivo xml
- Declaração de DTD
  - Útil para auto-completion
- Declaração das classes
- Usualmente possuem extensão .hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
[...]
</hibernate-mapping>
```

XML

- Recomendação da W3C para gerar linguagens de marcação para necessidades especiais.
  - Separação do conteúdo e da formatação
  - legibilidade tanto por humanos quanto por máquinas
  - possibilidade de criação de tags sem limitação
  - criação de arquivos para validação de estrutura
  - com seu uso pode-se interligar bancos de dados distintos
  - Simplicidade
  - XML concentra-se na estrutura da informação e não na sua aparência

XML - Exemplo

```
<?xml version="1.0" encoding="UTF-8"?>

<Receita nome="pão" tempo_de_preparo="5 minutos" tempo_de_cozimento="1 hora">

  <título>Pão simples</título>

  <ingrediente quantidade="3" unidade="xícaras">Farinha</ingrediente>
  <ingrediente quantidade="7" unidade="gramas">Fermento</ingrediente>
  <ingrediente quantidade="1.5" unidade="xícaras" estado="morna">Água</ingrediente>
  <ingrediente quantidade="1" unidade="colheres de chá">Sal</ingrediente>

  <Instruções>
   <passo>Misture todos os ingredientes, e dissolva bem.</passo>
   <passo>Cubra com um pano e deixe por uma hora em um local morno.</passo>
   <passo>Misture novamente, coloque numa bandeja e asse num forno.</passo>
  </Instruções>

</Receita>
```

XML – Referencias & Ferramentas

- Referencias
  - http://www.w3schools.com/xml/
  - http://www.w3.org/XML/

- Ferramentas
  - XMLSpy
  - Eclipse
  - Qualquer editor de texto

Arquivo Mapeamento – Ex POJO

```
package events;

import java.util.Date;

public class Event {
    private Long id;

    private String title;
    private Date date;

    public Event() {}

    public Long getId() {
        return id;
    }

    private void setId(Long id) {
        this.id = id;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

Mapeamento de Classe

- Indicação nome da Classe
- Indicação nome da Tabela

```
<hibernate-mapping>

    <class name="events.Event" table="EVENTS">

    </class>

</hibernate-mapping>
```

Mapeamento de Identificadores

- Indicação de PK
  - Classe
  - Tabela
- Escolha de gerador para geração de Chave

```
<hibernate-mapping>

    <class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
    </class>

</hibernate-mapping>
```

Tipos de Geradores de Chave

- increment
- identity
- sequence
- hilo
- seqhilo
- uuid
- guid
- **native**
- assigned
- select
- foreign

Mapeamento de Propriedades

- Indicação de nome, tipo e coluna
  - Se não explicitar coluna considera o nome como coluna
- date é palavra reservada em SQL, logo deve ser mapeada para EVENT_DATE

```
<hibernate-mapping>

    <class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

Mapeamento de Relacionamentos

- one-to-one
- one-to-many
- many-to-many

Outro Exemplo POJO - Person

```
package events;

public class Person {

    private Long id;
    private int age;
    private String firstname;
    private String lastname;

    public Person() {}

    // Accessor methods for all properties, private setter for 'id'

}
```

Mapeamento Person

```
<hibernate-mapping>

    <class name="events.Person" table="PERSON">
        <id name="id" column="PERSON_ID">
            <generator class="native"/>
        </id>
        <property name="age"/>
        <property name="firstname"/>
        <property name="lastname"/>
    </class>

</hibernate-mapping>
```

Mapeamento – one-to-one

- Inclusão de um coordenador único por evento

```
<hibernate-mapping>

   <class name="events.Event" table="EVENTS">
      <id name="id" column="EVENT_ID">
         <generator class="native"/>
      </id>
      <property name="date" type="timestamp" column="EVENT_DATE"/>
      <property name="title"/>
      <one-to-one name="coordinator" class="Person" />
   </class>

</hibernate-mapping>
```

Mapeamento – many-to-one

- Inclusão de um coordenador para vários eventos
  - Class é necessário??

```
<hibernate-mapping>

   <class name="events.Event" table="EVENTS">
      <id name="id" column="EVENT_ID">
         <generator class="native"/>
      </id>
      <property name="date" type="timestamp" column="EVENT_DATE"/>
      <property name="title"/>
      <many-to-one name=“coordinator” class=“Person”  column=“COORDINATOR_ID”/>
   </class>

</hibernate-mapping>
```

Mapeamento – uso de Sets

- one-to-many
- Um coordenador para vários eventos

```
<class name="events.Person" table="PERSON">
    <id name="id" column="PERSON_ID">
        <generator class="native"/>
    </id>
    <property name="age"/>
    <property name="firstname"/>
    <property name="lastname"/>

     <set name=“eventos">
         <key column=“EVENT_ID"/>
         <one-to-many class=" events.Event" />
     </set>

</class>
```

Mapeamento – uso de Sets

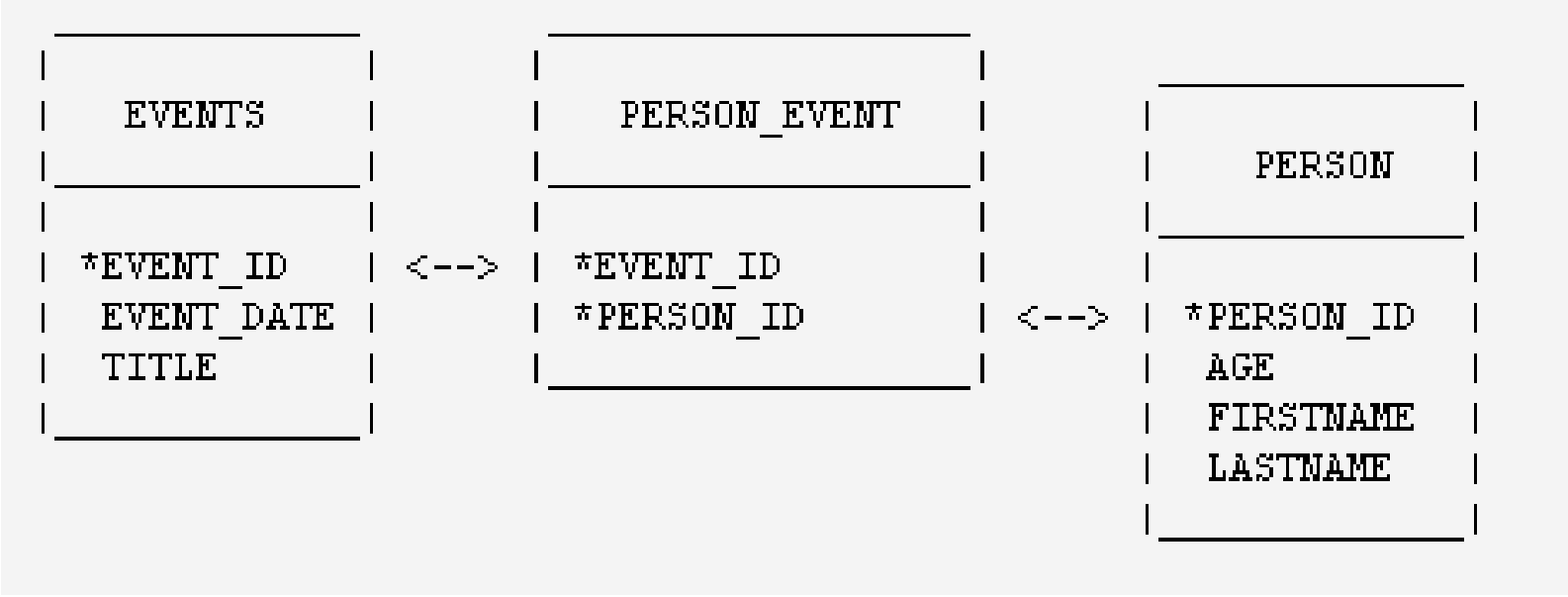
- many-to-many
- Inclusão de eventos para uma pessoa
- Tabela de associação

```
<class name="events.Person" table="PERSON">
    <id name="id" column="PERSON_ID">
        <generator class="native"/>
    </id>
    <property name="age"/>
    <property name="firstname"/>
    <property name="lastname"/>

    <set name="events" table="PERSON_EVENT">
        <key column="PERSON_ID"/>
        <many-to-many column="EVENT_ID" class="events.Event"/>
    </set>

</class>
```

Tabela de Associação



Sets Ordenados

- Atributo sort em set
- Fornecer um java.util.Comparator

```
<set name="comentarios"
sort="br.com.facio.cityshoes.util.PedidoComentarioComparator" >

            <key column="id_pedido"/>
             <one-to-many
class="br.com.facio.cityshoes.model.PedidoComentario"/>

</set>
```

Coleção de Valores

- Mapeamento de valores que não são entidades
- Ex. Endereço

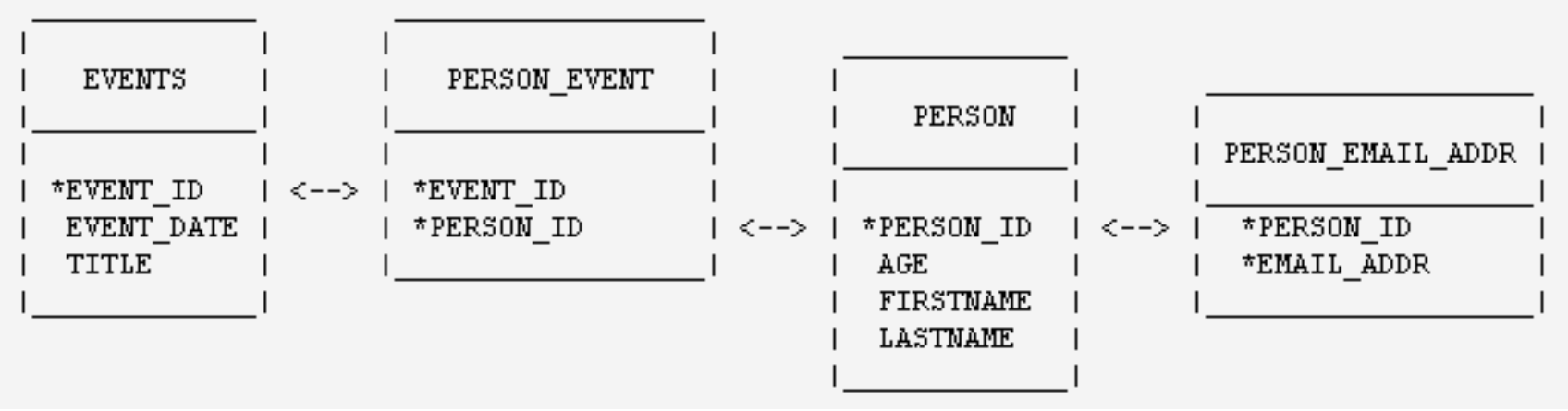
```
private Set emailAddresses = new HashSet();

public Set getEmailAddresses() {
    return emailAddresses;
}
public void setEmailAddresses(Set emailAddresses) {
    this.emailAddresses = emailAddresses;
}
```

```
<set name="emailAddresses" table="PERSON_EMAIL_ADDR">
    <key column="PERSON_ID"/>
    <element type="string" column="EMAIL_ADDR"/>
</set>
```

Coleção de Valores

- Equivalente a criação de um atributo composto com relacionamento de 1:n no modelo relacional



Heranças

- Estratégia de estrutura genérica não utilizada

- Três estratégias
  - Tabela por hierarquia
  - Tabela por sub-classe
  - Tabela por classe concreta

- Recurso
  - Polimorfismo implícito

Tabela por Hierarquia

- Uso de discriminador
- Ex. Payment

```
<class name="Payment" table="PAYMENT">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="native"/>
    </id>
    <discriminator column="PAYMENT_TYPE" type="string"/>
    <property name="amount" column="AMOUNT"/>
    ...
    <subclass name="CreditCardPayment" discriminator-value="CREDIT">
        <property name="creditCardType" column="CCTYPE"/>
        ...
    </subclass>
    <subclass name="CashPayment" discriminator-value="CASH">
        ...
    </subclass>
    <subclass name="ChequePayment" discriminator-value="CHEQUE">
        ...
    </subclass>
</class>
```

Tabela por sub-classe

- Quatro tabelas
- Três tabelas de sub-classes referenciam a tabela de super-classe

Tabela por sub-classe

```
<class name="Payment" table="PAYMENT">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="native"/>
    </id>
    <property name="amount" column="AMOUNT"/>
    ...
    <joined-subclass name="CreditCardPayment" table="CREDIT_PAYMENT">
        <key column="PAYMENT_ID"/>
        <property name="creditCardType" column="CCTYPE"/>
        ...
    </joined-subclass>
    <joined-subclass name="CashPayment" table="CASH_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </joined-subclass>
    <joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </joined-subclass>
</class>
```

Tabela por classe concreta

- Três tabelas para as sub-classes
- Cada tabela define colunas para todas propriedades, mesmo aquelas herdadas

Tabela por classe concreta

```
<class name="Payment">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="sequence"/>
    </id>
    <property name="amount" column="AMOUNT"/>
    ...
    <union-subclass name="CreditCardPayment" table="CREDIT_PAYMENT">
        <property name="creditCardType" column="CCTYPE"/>
        ...
    </union-subclass>
    <union-subclass name="CashPayment" table="CASH_PAYMENT">
        ...
    </union-subclass>
    <union-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        ...
    </union-subclass>
</class>
```

Tabela por classe concreta usando polimorfismo implícito

- Todas as tabelas são mapeadas redefinindo as propriedades em comum

Tabela por classe concreta usando polimorfismo implícito

```
<class name="CreditCardPayment" table="CREDIT_PAYMENT">
   <id name="id" type="long" column="CREDIT_PAYMENT_ID">
      <generator class="native"/>
   </id>
   <property name="amount" column="CREDIT_AMOUNT"/>
   ...
</class>
<class name="CashPayment" table="CASH_PAYMENT">
   <id name="id" type="long" column="CASH_PAYMENT_ID">
      <generator class="native"/>
   </id>
   <property name="amount" column="CASH_AMOUNT"/>
   ...
</class>
<class name="ChequePayment" table="CHEQUE_PAYMENT">
   <id name="id" type="long" column="CHEQUE_PAYMENT_ID">
      <generator class="native"/>
   </id>
   <property name="amount" column="CHEQUE_AMOUNT"/>
   ...
</class>
```

Hibernate – Trabalho com Objetos

Estados das Instancias

- Transient
  - Nunca associado a um contexto
  - Não possui identificador de persistência (PK)
- Persistent
  - Associada a um contexto
  - Possui um identificador de persistência (PK)
    - Possivelmente possui uma equivalência com a base
  - O Hibernate garante a consistência de objetos persistidos com aqueles em memória
- Detached
  - Já foram associados a uma sessão
  - Possuem identidade de persistência
    - Dados na base
  - Hibernate não garante a consistência dos objetos java “detacheds” com aqueles persistidos

Tornando um objeto persistente

- Basta inserir um objeto transiente ou detached em uma sessão
  - Método save
- Caso seja transiente um identificador é associado ao objeto

```
DomesticCat fritz = new DomesticCat();
fritz.setColor(Color.GINGER);
fritz.setSex('M');
fritz.setName("Fritz");
Long generatedId = (Long) sess.save(fritz);
```

- O usuário pode fornecer o identificador

```
DomesticCat pk = new DomesticCat();
pk.setColor(Color.TABBY);
pk.setSex('F');
pk.setName("PK");
pk.setKittens( new HashSet() );
pk.addKitten(fritz);
sess.save( pk, new Long(1234) );
```

Carregando um objeto

- Sessão fornece método para carregar objeto em memória
  - load
  - get

```
long id = 1234;
DomesticCat pk = (DomesticCat) sess.load( DomesticCat.class, new Long(id) );
```

- Load Permite carregar estado persistido em instância já existente

```
Cat cat = new DomesticCat();
// load pk's state into cat
sess.load( cat, new Long(pkId) );
Set kittens = cat.getKittens();
```

load() e get()

- Load
  - Lança exceção caso não encontre o objeto na base de dados
- Get
  - Retorna nulo caso não encontre o objeto

```
Cat cat = (Cat) sess.get(Cat.class, id);
if (cat==null) {
    cat = new Cat();
    sess.save(cat, id);
}
return cat;
```

Buscando Objetos

- HQL
  - Hibernate Query Language
- QBC
  - Query by Criteria
- QBE
  - Query by Example
- SQL

Interface de Query

- org.hibernate.Query
- Obtida a partir da Session
- .list() retorna o resultado da Query

Exemplos de Query

```
List cats = session.createQuery(
    "from Cat as cat where cat.birthdate < ?")
    .setDate(0, date)
    .list();

List mothers = session.createQuery(
    "select mother from Cat as cat join cat.mother as mother where cat.name = ?")
    .setString(0, name)
    .list();

List kittens = session.createQuery(
    "from Cat as cat where cat.mother = ?")
    .setEntity(0, pk)
    .list();

Cat mother = (Cat) session.createQuery(
    "select cat.mother from Cat as cat where cat = ?")
    .setEntity(0, izi)
    .uniqueResult();]]

Query mothersWithKittens = (Cat) session.createQuery(
    "select mother from Cat as mother left join fetch mother.kittens");
Set uniqueMothers = new HashSet(mothersWithKittens.list());
```

Iteração de resultados

- list()
  - Aloca todos os resultados da query em memória
- iterate()
  - Acessa a base a cada chamada de next()
  - Útil quando espera-se resultados existentes em cache

```
// fetch ids
Iterator iter = sess.createQuery("from eg.Qux q order by q.likeliness").iterate();
while ( iter.hasNext() ) {
    Qux qux = (Qux) iter.next();  // fetch the object
    // something we couldnt express in the query
    if ( qux.calculateComplicatedAlgorithm() ) {
        // delete the current instance
        iter.remove();
        // dont need to process the rest
        break;
    }
}
```

Queries que retornam tuplas

- Cada tupla é interpretada como um array
  - Não precisa de Cast?!?!?

```
Iterator kittensAndMothers = sess.createQuery(
        "select kitten, mother from Cat kitten join kitten.mother mother")
        .list()
        .iterator();

while ( kittensAndMothers.hasNext() ) {
    Object[] tuple = (Object[]) kittensAndMothers.next();
    Cat kitten  = tuple[0];
    Cat mother  = tuple[1];
    ....
}
```

Queries com parâmetros

- "Binding Values"
  - ?
  - :param
- :param
  - Não são sensíveis a ordem de ocorrência
  - Podem ocorrer diversas vezes em uma única query
  - São auto-documentáveis

```
//named parameter (preferred)
Query q = sess.createQuery("from DomesticCat cat where cat.name = :name");
q.setString("name", "Fritz");
Iterator cats = q.iterate();
```

```
//positional parameter
Query q = sess.createQuery("from DomesticCat cat where cat.name = ?");
q.setString(0, "Izi");
Iterator cats = q.iterate();
```

Paginação

- Especificar o número máximo de ocorrencias
- Primeira linha a ser retornada
  - ???

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

Criteria Queries

- API orientada a objetos para busca

```
Criteria crit = session.createCriteria(Cat.class);
crit.add( Expression.eq( "color", eg.Color.BLACK ) );
crit.setMaxResults(10);
List cats = crit.list();
```

Hibernate – Sessões e Transações

Escopo - SessionFactory

- SessionFactory
  - Alto custo de criação
  - Geralmente uma única por aplicação
  - Obtida a partir de uma instância de Configuration

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {  return sessionFactory;  }
}
```

Escopo – Session e Transaction

- Session
  - Baixo custo de criação
  - Non-threadsafe
  - Utilizada para
    - Requisição
    - Conversações
    - Unidade de trabalho
  - Apenas utiliza uma conexão com a base caso seja realmente necessário
- Transaction
  - Devem ser o quanto menores possível
    - Redução de lock na base de dados
  - Conexão com a base de dados

Padrões e anti-padrões

- Padrões
  - Session-per-request
    - A cada requisição do usuário ao servidor uma sessão é aberta
    - Diversas formas de implementação
      - ServletFilter
      - AOP
      - Proxy
  - Open-Session-in-View
    - Manter a sessão aberta até “renderizar” a camada de visualização
- Anti-padrões
  - Session-per-operation
    - Abrir uma sessão/transação para cada operação com a base de dados em uma única thread.

Exemplo

```
public static void main(String[] args) {
        EventManager mgr = new EventManager();
        if (args[0].equals("store")) {
            mgr.createAndStoreEvent("My Event", new Date());
        }
        HibernateUtil.getSessionFactory().close();
    }

    private void createAndStoreEvent(String title, Date theDate) {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Event theEvent = new Event();
        theEvent.setTitle(title);
        theEvent.setDate(theDate);
        session.save(theEvent);
        session.getTransaction().commit();
    }
```

Referencias

- Documentação Hibernate
  - http://www.hibernate.org/hib_docs/v3/reference/en/html/

- Mapping Objects to Relational Databases: O/R Mapping In Detail
  - http://www.agiledata.org/essays/mappingObjects.html

Exercícios

- Trabalhar a “auto-didática”
  - Ler artigo de referência
  - Ler capítulos da documentação do hibernate
    - 1
    - 2
    - 3
    - 4
    - 10
    - 11
- Desenvolver aplicação do próximo slide e/ou outra aplicação que considere mais interessante
  - usando o hibernate, é claro

Aplicação para controle de gastos

- Suponha uma aplicação para fazer o controle de gastos.
- Nela seria possível cadastrar entradas de credito ou débito que determinado indivíduo realizou.
- Cada entrada possui um valor, uma data e uma categoria.
- Deve ser possível
  - Registrar categorias
  - Verificar quanto foi gasto em determinada categoria em determinado intervalo de tempo.
  - Verificar qual foi a percentagem que determinada categoria gastou em determinado intervalo de tempo.
- Não se preocupe com
  - Interface
  - Controle de usuários