



1

# Giới Thiệu Hibernate

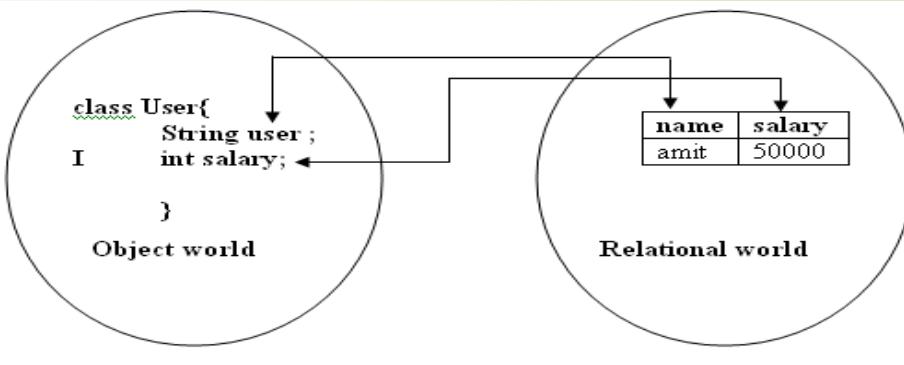
Trình bày : Nguyễn Thanh Tùng

# NỘI DUNG TRÌNH BÀY

- I. ORM và Hibernate
- II. Xây dựng ứng dụng nhỏ sử dụng kết nối Hibernate
- III. Thiết lập ánh xạ các mối quan hệ thường dùng
- IV. Demo ánh xạ tự động bằng Netbeans tool
- V. Vòng đời đối tượng trong Hibernate
- VI. Session và transaction
- VII.Cơ chế Fetching
- VIII.HQL

# I. ORM là gì ?

- ▶ ORM (Object – Relational – Mapping) là một kỹ thuật lập trình cho việc chuyển đổi dữ liệu giữa CSDL quan hệ và các ngôn ngữ lập trình hướng đối tượng(Java, C#, ...)



- ▶ Ưu điểm của ORM so với kiểu kết nối CSDL cũ như JDBC (Java DataBase Connectivity):
  - ▶ Ân các chi tiết của các truy vấn SQL (ORM thực hiện truy vấn trên các đối tượng)
  - ▶ Lập trình viên không cần biết chi tiết hiện thực database bên dưới
  - ▶ Giảm lượng code cần viết (đối với ứng dụng lớn)
  - ▶ ...

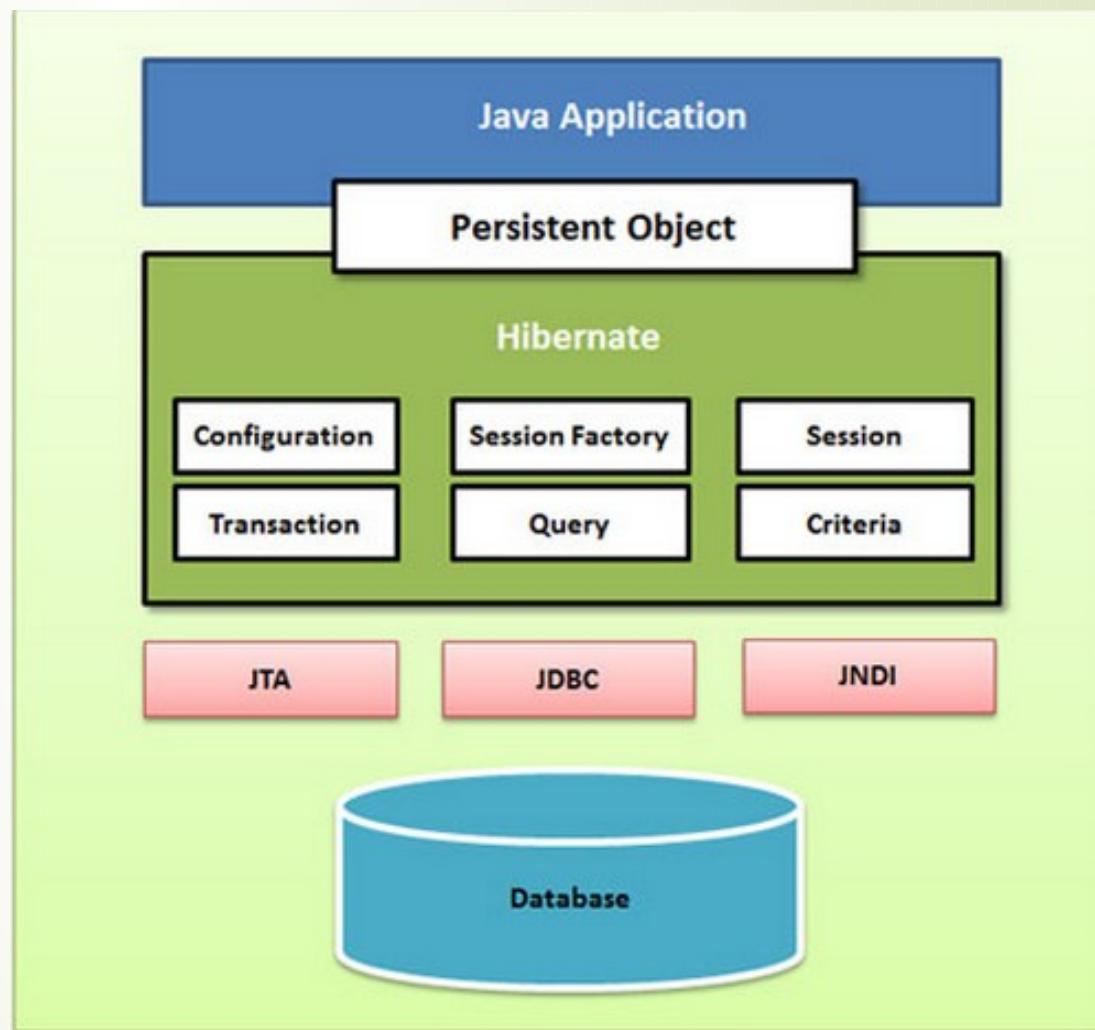
# Hibernate là gì ?

- ▶ Là 1 kỹ thuật ORM trên Java



- ▶ Là mã nguồn mở
- ▶ Hỗ trợ các API để lưu trữ và lấy các đối tượng Java trực tiếp từ database
- ▶ Hỗ trợ hầu hết các hệ thống cơ sở dữ liệu quan hệ: MySQL, Oracle, Microsoft SQL, ...

# Cấu trúc ứng dụng sử dụng Hibernate



Cấu trúc ứng dụng Hibernate  
với những lớp quan trọng

# Thành phần JDBC, JTA, JNDI

- ▶ JDBC, JTA(Java Transaction API), JNDI( Java Naming and Directory Interface) là các API có sẵn của Java.
- ▶ JDBC cung cấp một mức trừu tượng cơ sở của các chức năng thường dùng trên CSDL quan hệ, cho phép hibernate kết nối đến bất kì CSDL nào được hỗ trợ bởi JDBC driver.
- ▶ JTA và JNDI cho phép Hibernate được tích hợp với J2EE application servers.

# Đối tượng Configuration

- ▶ Đối tượng Hibernate đầu tiên được tạo trong bất kỳ ứng dụng Hibernate, và được tạo chỉ một lần trong suốt quá trình ứng dụng chạy.
- ▶ Gồm 2 thành phần chính :
  - ▶ Database Connection : Được lấy thông qua 2 file : hibernate.properties và hibernate.cfg.xml
  - ▶ Class Mapping Setup: Thành phần này tạo các kết nối giữa các lớp Java và các bảng trong CSDL quan hệ. Ex: User.hbm.xml

# Đối tượng SessionFactory, Session, Transaction

- ▶ SessionFactory:
  - ▶ Được tạo ra bởi đối tượng Configuration
  - ▶ Sử dụng tạo ra đối tượng Session
  - ▶ Được khởi tạo chỉ một lần và được sử dụng cho nhiều thread của ứng dụng
- ▶ Session
  - ▶ Được sử dụng để lấy một kết nối vật lý với 1 CSDL
  - ▶ Lưu trữ và lấy dữ liệu thông qua Session
  - ▶ Session chỉ được sử dụng cho 1 thread
- ▶ Transaction
  - ▶ Đại diện cho 1 đơn vị làm việc với CSDL

# Đối tượng Query và Criteria

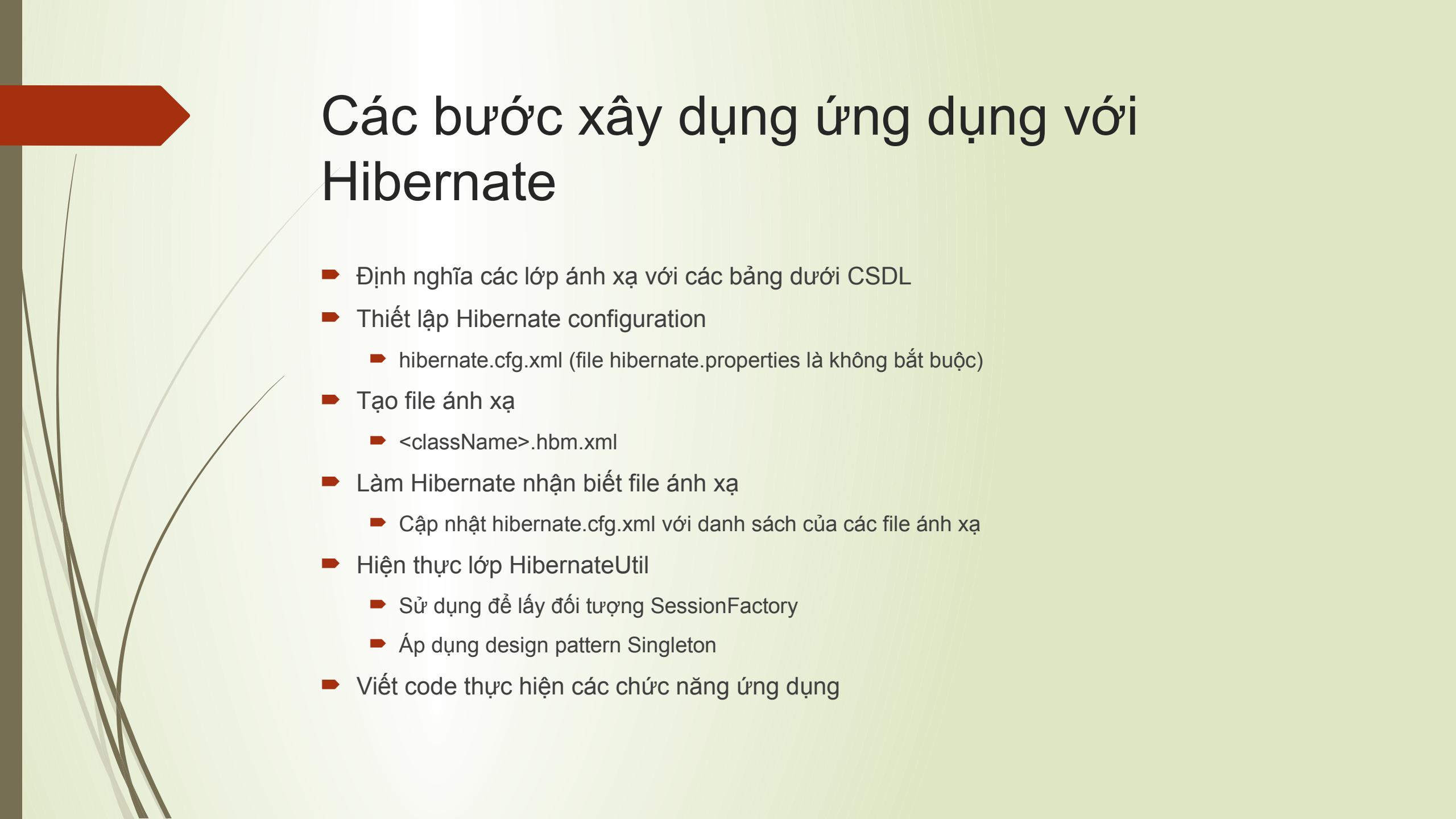
- ▶ **Query:**
  - ▶ Sử dụng SQL hoặc HQL (Hibernate Query Language) để lấy dữ liệu và tạo các đối tượng
  - ▶ Thực hiện truy vấn trên các đối tượng
- ▶ **Criteria**
  - ▶ Sử dụng để tạo và thực hiện các truy vấn , bộ lọc theo hướng đối tượng để lấy các đối tượng.

## II. Ứng dụng nhỏ sử dụng Hibernate

- ▶ CSDL : 1 bảng employees

employees	
!	id INT(11)
◆	age INT(11)
◆	first VARCHAR(255)
◆	last VARCHAR(255)
Indexes	

- ▶ Ứng dụng sẽ lấy tất cả dữ liệu employee dưới CSDL và hiển thị.
- ▶ Hibernate sử dụng 2 kiểu ánh xạ giữa Java class và bảng là : file XML và Annotation. Trong slide sẽ sử dụng kiểu file XML.



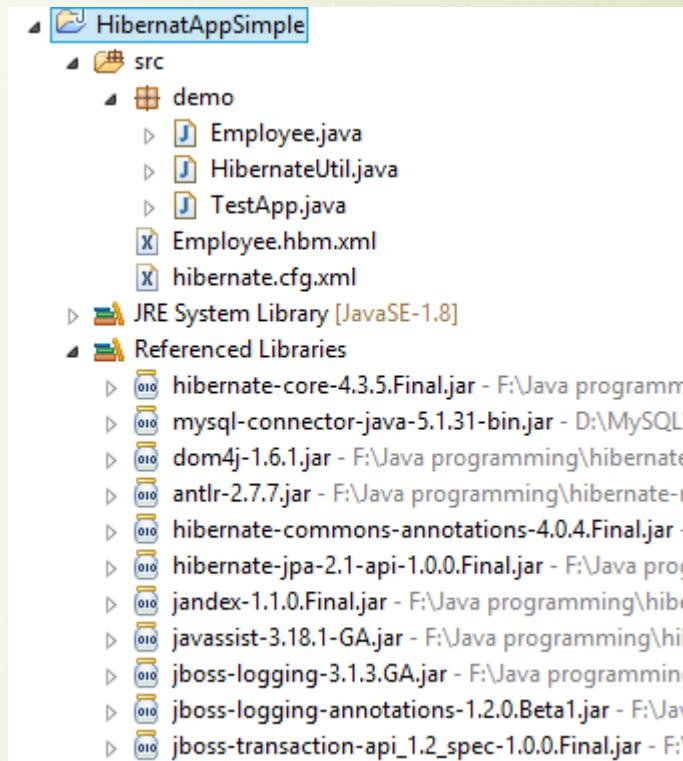
# Các bước xây dựng ứng dụng với Hibernate

- ▶ Định nghĩa các lớp ánh xạ với các bảng dưới CSDL
- ▶ Thiết lập Hibernate configuration
  - ▶ hibernate.cfg.xml (file hibernate.properties là không bắt buộc)
- ▶ Tạo file ánh xạ
  - ▶ <className>.hbm.xml
- ▶ Làm Hibernate nhận biết file ánh xạ
  - ▶ Cập nhật hibernate.cfg.xml với danh sách của các file ánh xạ
- ▶ Hiện thực lớp HibernateUtil
  - ▶ Sử dụng để lấy đối tượng SessionFactory
  - ▶ Áp dụng design pattern Singleton
- ▶ Viết code thực hiện các chức năng ứng dụng

# Cấu trúc chương trình

- ▶ Thêm các file .jar :

- ▶ Tất cả các file trong thư mục \hibernate-release-4.x.x.Final\lib\required
- ▶ mysql-connector-java-5.x.x-bin.jar



# Lớp Employee

```
package demo;

public class Employee {
    private long id;
    private int age;
    private String first;
    private String last;

    public Employee() {

    }

    //getter and setter
}
```

# hibernate.cfg.xml

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

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://.hibernate.sourceforge.net/
  hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    ...
  </session-factory>
<hibernate-configuration>
```

Mô tả các cấu hình cho hibernate

# Hibernate.cfg.xml

```
<session-factory>
    <!-- Database connection settings -->
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost/testdb</property>
    <property name="hibernate.connection.username">testuser</property>
    <property name="hibernate.connection.password">testpass</property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>

    <!-- Dialect is required to let Hibernate know the Database Type, MySQL, Oracle etc -->
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>
</session-factory>
```

# Configuration Hibernate

- ▶ Hibernate sẽ tìm kiếm và cấu hình theo thứ tự :
    - ▶ hibernate.properties (khi 'new Configuration()' được gọi)
    - ▶ hibernate.cfg.xml (khi phương thức 'configure()' của Configuration được gọi)
  - ▶ Configuration configuration = new Configuration()  
    .configure("hibernate.cfg.xml");
- Load file Hibernate.properties**
- Load file Hibernate.cfg.xml**

# File ánh xạ Employee.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>
    <class>
        ...
    </class>
</hibernate-mapping>
```

Mô tả các thuộc tính của class

# File ánh xạ Employee.hbm.xml

```
<class name="demo.Employee" table="employees">
    <id name="id">
        <column name="id" not-null="true"/>
        <generator class="native" />
    </id>
    <property name="age" type="integer" not-null="true">
        <column name="age" />
    </property>
    <property name="first" type="string">
        <column name="first" />
    </property>
    <property name="last" type="string">
        <column name="last" />
    </property>
</class>
```

# HibernateUtil

- ▶ Lớp tiện ích để xây dựng và đạt được Hibernate SessionFactory
  - ▶ Được đề nghị sử dụng bởi Hibernate.org
- ▶ SessionFactory là thread-safe
  - ▶ Duy nhất cho toàn bộ ứng dụng
- ▶ Được sử dụng để xây dựng Hibernate Sessions

# HibernateUtil

```
public class HibernateUtil {  
    private static final SessionFactory sessionFactory;  
  
    private static SessionFactory buildSessionFactory() {  
        try {  
            Configuration configuration = new Configuration().configure("hibernate.cfg.xml");  
            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()  
                .applySettings(configuration.getProperties()).build();  
            sessionFactory = configuration.buildSessionFactory(serviceRegistry);  
        }  
        catch (Throwable ex) {  
            System.err.println("Initial SessionFactory creation failed." + ex);  
            throw new ExceptionInInitializerError(ex);  
        }  
    }  
  
    public static SessionFactory getSessionFactory() {  
        return sessionFactory;  
    }  
}
```

# Sử dụng Session để lấy ListEmployee

```
private static List<Employee> getAllEmployee() {  
    List<Employee> listEmployees = null;  
  
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
  
    listEmployees = session.createQuery("from Employee").list();  
  
    session.getTransaction().commit();  
  
    return listEmployees;  
}
```

# Output của chương trình

► Hàm main:

```
public static void main(String[] args) {  
    List<Employee> listEmployee = ...  
  
    System.out.println("\n\n\n    ID  
    System.out.println("-----");  
    for (Employee employee : listEmployee) {  
        System.out.println(String.format("%6d%6d%14s%13s",  
            employee.getId(), employee.getAge(),  
            employee.getFirst(), employee.getLast()));  
    }  
}
```

ID	Age	First	Last
101	25	Mahnaz	Fatma
102	30	Zaid	Khan
103	28	Sumit	Mittal



### III. Cách thiết lập ánh xạ các mối quan hệ thường dùng

- ▶ Hướng của mối quan hệ
- ▶ Các kiểu Hibernate collection thường dùng
- ▶ Quan hệ hai chiều
  - ▶ One-to-Many / Many-to-One
  - ▶ One-to-One
  - ▶ Many-to-Many

# Hướng của quan hệ

- ▶ Một chiều :
  - ▶ Chỉ có thể đi qua các đối tượng từ một mặt của quan hệ
  - ▶ Ví dụ: Account : Transaction
    - ▶ Cho một đối tượng Account, có thể đạt được các đối tượng Transaction liên quan
    - ▶ Cho một đối tượng Transaction, không thể đạt được đối tượng Account liên quan
- ▶ Hai chiều :
  - ▶ Có thể đi qua các đối tượng từ hai mặt của quan hệ
  - ▶ Ví dụ: Account : Transaction
    - ▶ Cho một đối tượng Account, có thể đạt được các đối tượng Transaction liên quan
    - ▶ Cho một đối tượng Transaction, có thể đạt được đối tượng Account liên quan.
- ▶ Trong bài giới thiệu về quan hệ hai chiều (được sử dụng phổ biến hơn).

# Các kiểu Hibernate collection thường dùng

- ▶ <set>
  - ▶ Không chứa các phần tử trùng nhau
  - ▶ Có thứ tự/không thứ tự
- ▶ <map>
  - ▶ Không chứa giá trị trùng nhau
  - ▶ Có thứ tự/không thứ tự, cần giá trị cho thuộc tính *key* và *column*
- ▶ <list>
  - ▶ Có thứ tự, cần một index column trên bảng đối tượng tham chiếu

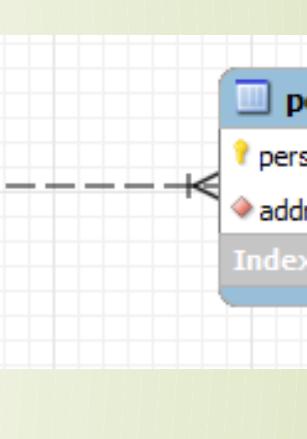
# One-to-Many / Many-to-One

```
package demo;

import java.util.Set;

public class Address {
    private int id;
    private Set<Person> people;

    //getter, setter and constructor
}
```



```
package demo;

public class Person {
    private int id;
    private Address address;

    //getter, setter and constructor
}
```

```
<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
    <set name="people" >
        <key column="addressId"/>
        <one-to-many class="Person"/>
    </set>
</class>
```

Address.hbm.xml

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <many-to-one name="address"
        column="addressId"
        not-null="true"/>
</class>
```

Person.hbm.xml

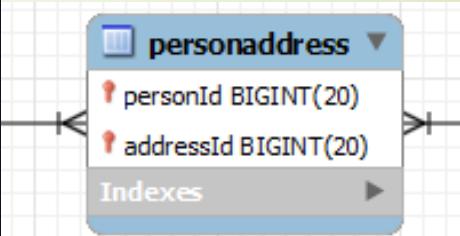
# Many-to-Many

```
package demo;

import java.util.Set;

public class Person {
    private int id;
    private Set<Address> addresses;

    //getter, setter and constructor
}
```



```
package demo;

import java.util.Set;

public class Address {
    private int id;
    private Set<Person> people;

    //getter, setter and constructor
}
```

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses" table="PersonAddress">
        <key column="personId"/>
        <many-to-many column="addressId"
                      class="Address"/>
    </set>
</class>
```

Address.hbm.xml

```
<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
    <set name="people" inverse="true"
          table="PersonAddress">
        <key column="addressId"/>
        <many-to-many column="personId"
                      class="Person"/>
    </set>
</class>
```

Person.hbm.xml

# One-to-One

```
package demo;

public class Address {
    private int id;
    private Person person;

    //getter, setter and constructor
}
```

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <one-to-one name="address"/>
</class>
```

Address.hbm.xml

```
package demo;

public class Person {
    private int id;
    private Address address;

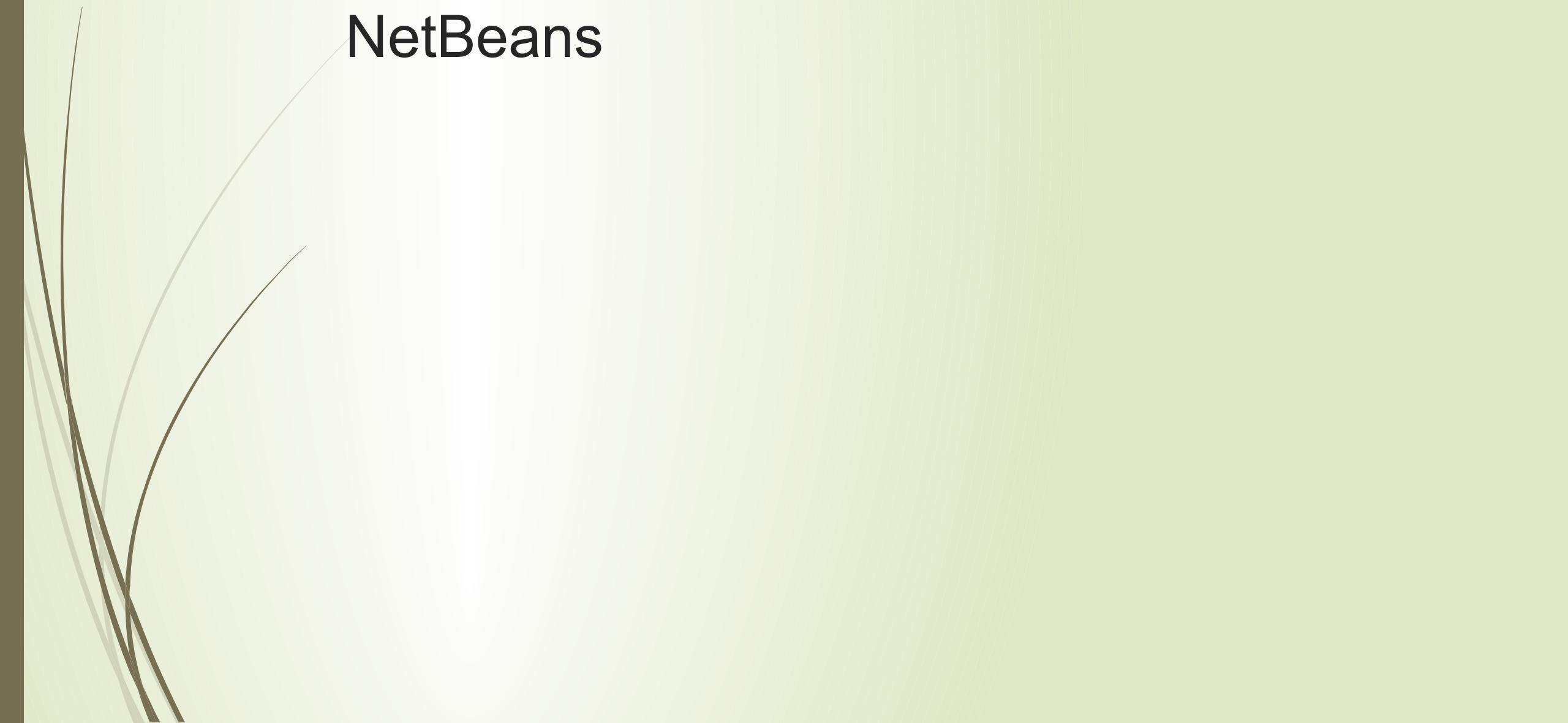
    //getter, setter and constructor
}
```

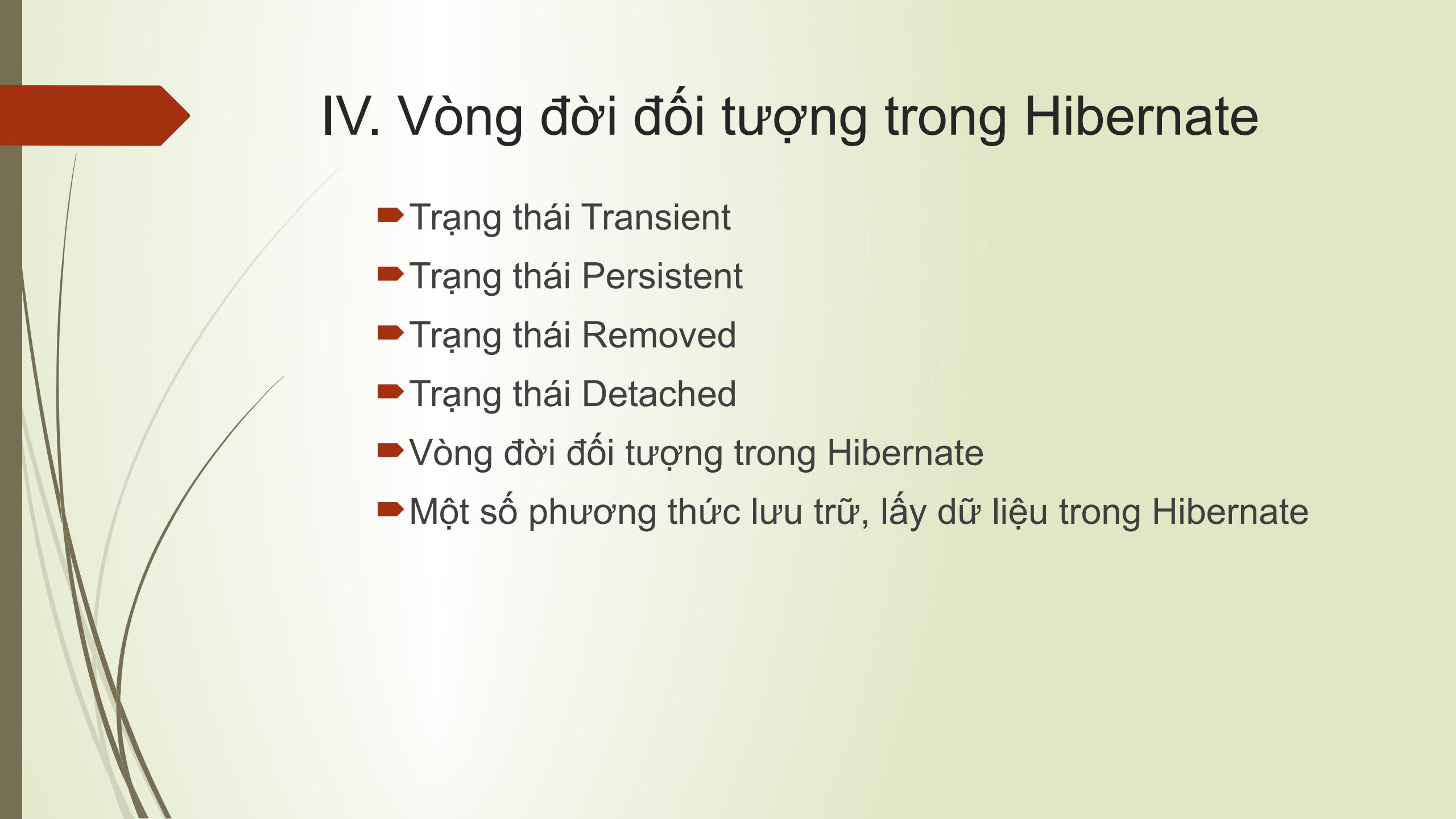
```
<class name="Address">
    <id name="id" column="personId">
        <generator class="foreign">
            <param name="property">
                person
            </param>
        </generator>
    </id>
    <one-to-one name="person"
        constrained="true"/>
</class>
```

Person.hbm.xml



# Demo ánh xạ tự động hibernate bằng NetBeans





## IV. Vòng đời đối tượng trong Hibernate

- ▶ Trạng thái Transient
- ▶ Trạng thái Persistent
- ▶ Trạng thái Removed
- ▶ Trạng thái Detached
- ▶ Vòng đời đối tượng trong Hibernate
- ▶ Một số phương thức lưu trữ, lấy dữ liệu trong Hibernate

## Trạng thái Transient

- ▶ Tất cả các đối tượng khi mới được khởi tạo có trạng thái Transient:
  - ▶ Account account = new Account();
  - ▶ account là một đối tượng Transient
- ▶ Hibernate không biết các đối tượng này
- ▶ Không liên quan với dòng trong CSDL
- ▶ Đối tượng này sẽ bị mất khi không được tham chiếu.

# Trạng thái Persistent

- ▶ Hibernate biết và quản lý đối tượng này
- ▶ Đối tượng có thể là :
  - ▶ Lấy một đối tượng đã tồn tại trong CSDL
  - ▶ Một đối tượng có trạng thái Transient đã được lưu
- ▶ Thay đổi các đối tượng có trạng thái Persistent được tự động lưu xuống CSDL mà không cần gọi các phương thức thực hiện
- ▶ Các đối tượng được đặt làm Persistent khi gọi các phương thức :
  - ▶ `session.save(account); session.lock(account);`
  - ▶ `session.update(account); session.merge(account);`

# Trạng thái Persistent

```
Session session = SessionFactory.getCurrentSession();
session.beginTransaction();
// trạng thái ‘transient’– Hibernate không biết sự tồn tại của đối tượng này
Account account new Account();
// Chuyển đổi thành trạng thái ‘persistent’. Hibernate bây giờ
// biết đối tượng này và quản lý nó
session.saveOrUpdate(account);
// Sự thay đổi của đối tượng sẽ tự động lưu bởi vì đối tượng
// bây giờ có trạng thái Persistent
account.setBalance(500);
// commit the transaction
session.getTransaction().commit();
```

## Trạng thái Removed

- ▶ Một đối tượng có trạng thái Persistent bị xóa khỏi CSDL
  - ▶ `session.delete(account);`
- ▶ Đối tượng Java có thể vẫn tồn tại nhưng nó được bỏ qua bởi Hibernate
- ▶ Bất kì thay đổi nào sẽ không thay đổi trong CSDL

# Trang thái Removed

```
Session session = SessionFactory.getCurrentSession();
session.beginTransaction();
// Lấy account với id 1. account được trả về có trạng thái 'persistent'
Account account = session.get(Account.class, 1);
// Chuyển sang trạng thái 'removed'. Hibernate sẽ xóa một bản ghi
// trong CSDL, và không quản lý đối tượng này.
session.delete(account);
// Sự thay đổi được bỏ qua bởi Hibernate
account.setBalance(500);
// commit the transaction
session.getTransaction().commit();
// Chú ý đối tượng Java vẫn sống ,mặc dù bị xóa khỏi CSDL.
account.setBalance(1000);
```

# Trạng thái Detached

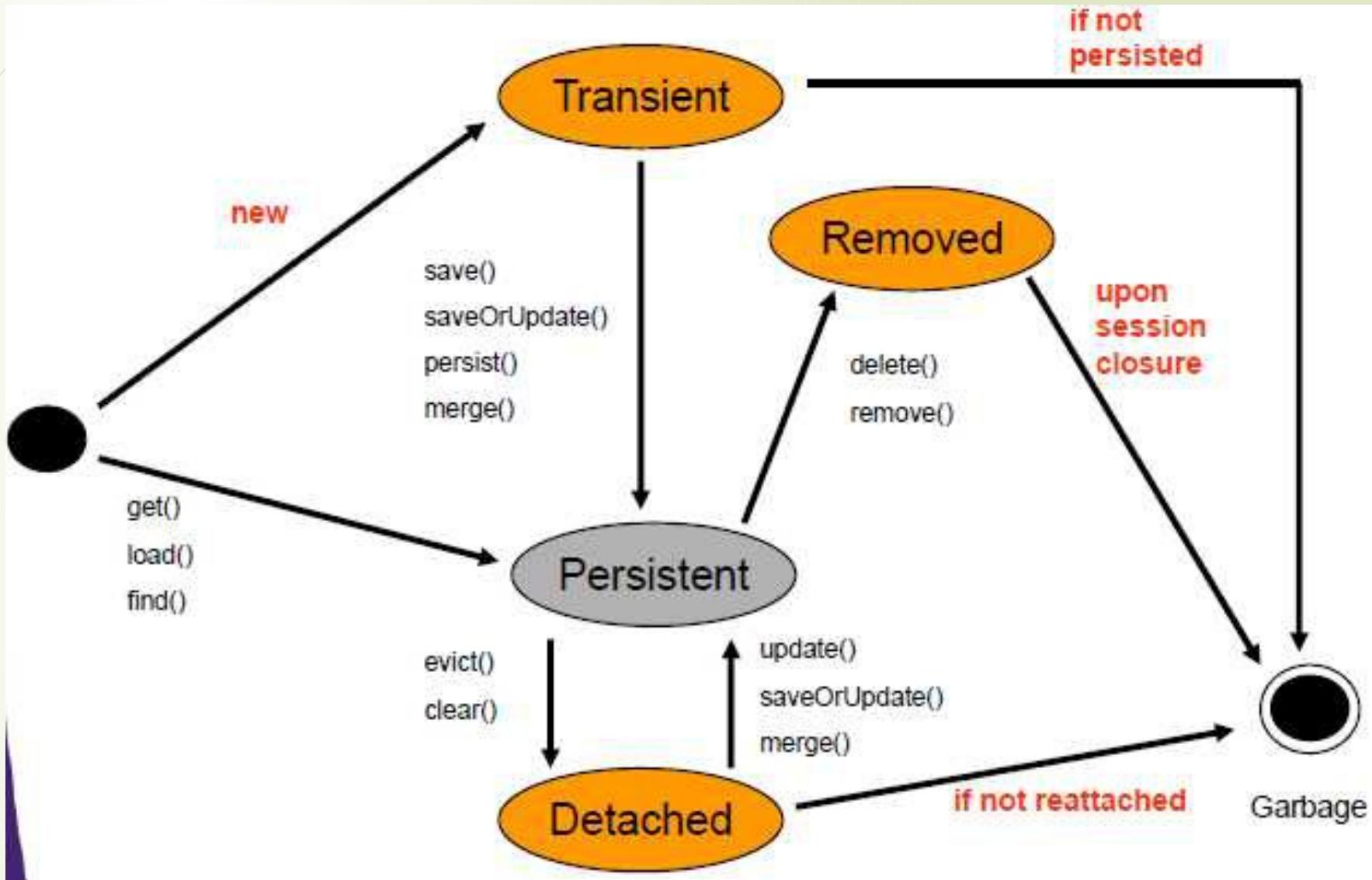
- ▶ session.close() thay đổi trạng thái đối tượng từ Persistent thành Detached
- ▶ Vẫn đại diện cho một dòng trong CSDL
- ▶ Không còn được quản lý bởi Hibernate
  - ▶ Sự thay đổi của đối tượng không được lưu trong CSDL
- ▶ Có thể reattached, đưa về trạng thái Persistent và đối tượng này được lưu vào CSDL
  - ▶ update();
  - ▶ merge();
  - ▶ lock(); // reattaches, nhưng không lưu thay đổi

# Trạng thái Detached

```
Session session1 = SessionFactory.getCurrentSession();
session.beginTransaction();
// Lấy account với id 1. account được trả về có trạng thái 'persistent'
Account account = session1.get(Account.class, 1);
// chuyển thanh trạng thái 'detached'. Hibernate không còn quản lý đối tượng này
session1.close();
// Sự thay đổi được bỏ qua bởi Hibernate từ khi nó có trạng thái 'detached'
// ,Nhưng account vẫn đại diện một dòng trong CSDL
account.setBalance(500);
// re-attach đối tượng bằng cách mở một session, gọi phương thức update() và sự //thay đổi trên đối tượng
được cập nhật.

Session session2 = SessionFactory.getCurrentSession();
session.beginTransaction();
session2.update(account);
// commit the transaction
session2.getTransaction().commit();
```

# Vòng đời đối tượng trong Hibernate



# Một số phương thức lưu trữ đối tượng

- ▶ `session.save(Object o)`
  - ▶ Thực hiện các câu lệnh để tạo một dòng mới trong CSDL
- ▶ `session.update(Object o)`
  - ▶ Thực hiện các câu lệnh để thay đổi một dòng đã tồn tại trong CSDL

# Một số phương thức lưu trữ đối tượng

- ▶ `session.saveOrUpdate(Object o)`
  - ▶ Xác định gọi phương thức thích hợp **save** or **update**
  
- ▶ `session.merge(Object o)`
  - ▶ Lấy tất cả các bản sao của đối tượng **o** hiện có trong bộ nhớ gộp lại (*merge*) tất cả các thay đổi vào trong một đối tượng Và thực hiện lưu xuống CSDL

# Một số phương thức lấy đối tượng

- ▶ `session.get(Object.class, Identifier)`
  - ▶ Gọi ngay lập tức một câu select để lấy đối tượng từ CSDL, một đối tượng được trả về đại diện cho một dòng trong CSDL.
  - ▶ Nếu không tìm thấy dữ liệu, trả về **null**
- ▶ `session.load(Object.class, Identifier)`
  - ▶ Nó luôn trả về một “**proxy**” Mà chưa thực hiện lấy dữ liệu từ CSDL.
  - ▶ Trong Hibernate , **proxy** là một đối tượng có giá trị ID, các thuộc tính chưa được khởi tạo.
  - ▶ Nếu không tìm thấy ,nó đưa ra một **ObjectNotFoundException**
  - ▶ Nó gọi đến CSDL khi đối tượng thực hiện các phương thức : `getxxx`

# Một số phương thức lấy đối tượng

- ▶ `session.lock(Object, LockMode)`
  - ▶ Reattaches một đối tượng có trạng thái detached với một session mà không cập nhật sự thay đổi.
- ▶ `session.refresh(Object)`
  - ▶ Lấy phiên bản mới nhất của đối tượng trong CSDL

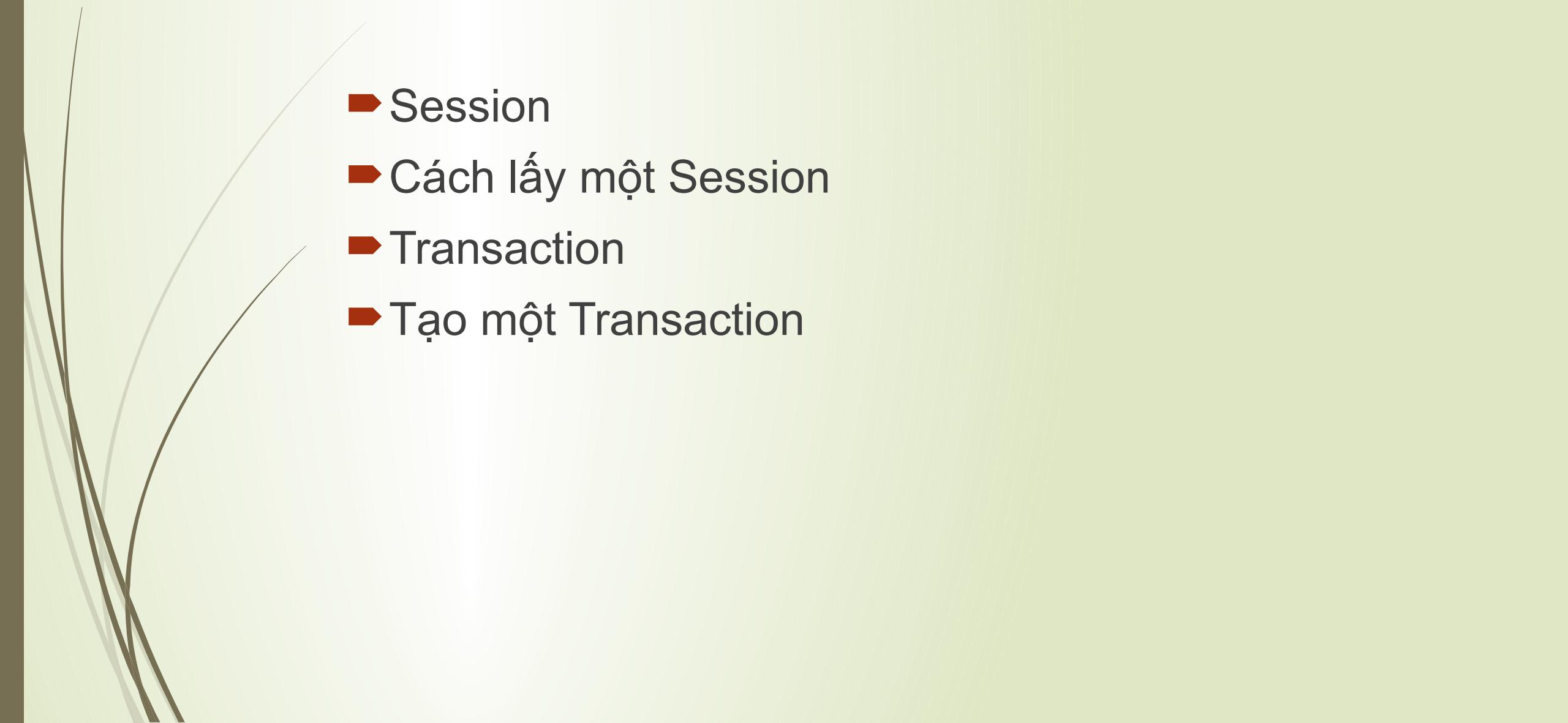
## Các phương thức khác

- ▶ `session.delete(Object)`
  - ▶ Thực hiện các câu lệnh để xóa đối tượng trong CSDL

▶ ...

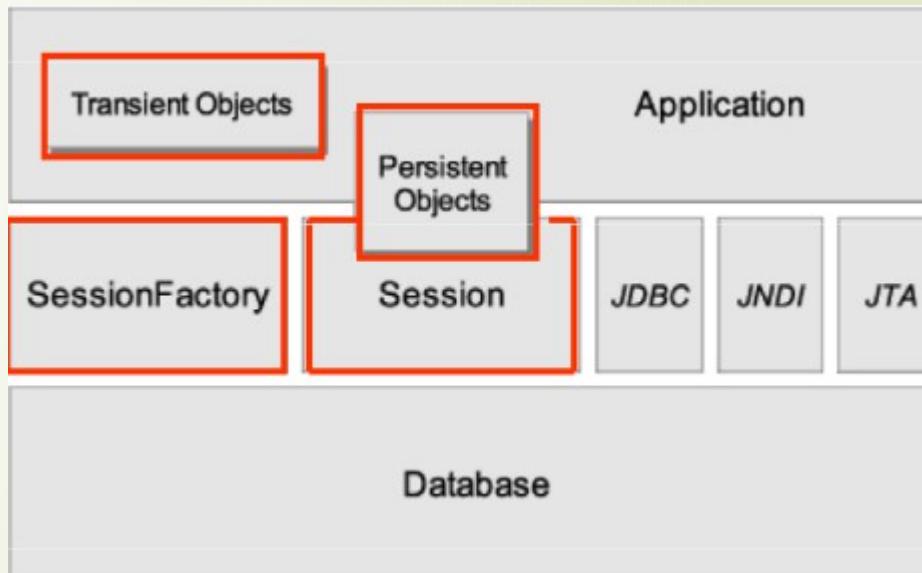


## V. Session và transaction

- ▶ Session
  - ▶ Cách lấy một Session
  - ▶ Transaction
  - ▶ Tạo một Transaction
- 

# Session trong hibernate

- ▶ A Session được sử dụng để lấy một kết nối vật lý đến CSDL.
- ▶ Một thread đơn không được chia sẻ giữa các đối tượng
- ▶ Chứa một số API thao thác với CSDL



# Đạt được một Session

- ▶ `SessionFactory.getCurrentSession()`
  - ▶ Lấy một Session đã tồn tại
  - ▶ Nếu không tồn tại, tạo một Session mới và lưu lại
- ▶ Tự động thực hiện các truy vấn dưới CSDL và đóng Session khi:
  - ▶ `transaction.commit` được gọi
- ▶ `SessionFactory.openSession()`
  - ▶ Luôn luôn đạt được một Session mới

# Transaction in Hibernate

- ▶ Đại diện một đơn vị làm việc
  - ▶ Tất cả các bước trong Transaction thành công -> commit()
  - ▶ Một bước thất bại -> rollback()
- ▶ Ví dụ: Một giao dịch tiền
  - ▶ Bước 1: tài khoản A gửi 500.000 đ
  - ▶ Bước 2: Tài khoản B nhận 500.000 đ từ A
- ▶ Trong Hibernate, các truy xuất đến CSDL nên nằm trong một Transaction.

# Ví dụ thực hiện một Transaction

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
Transaction tx = null;
try {
    tx = session.beginTransaction();
    // do some work
    ...
    tx.commit();
}
catch (Exception e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
}
```

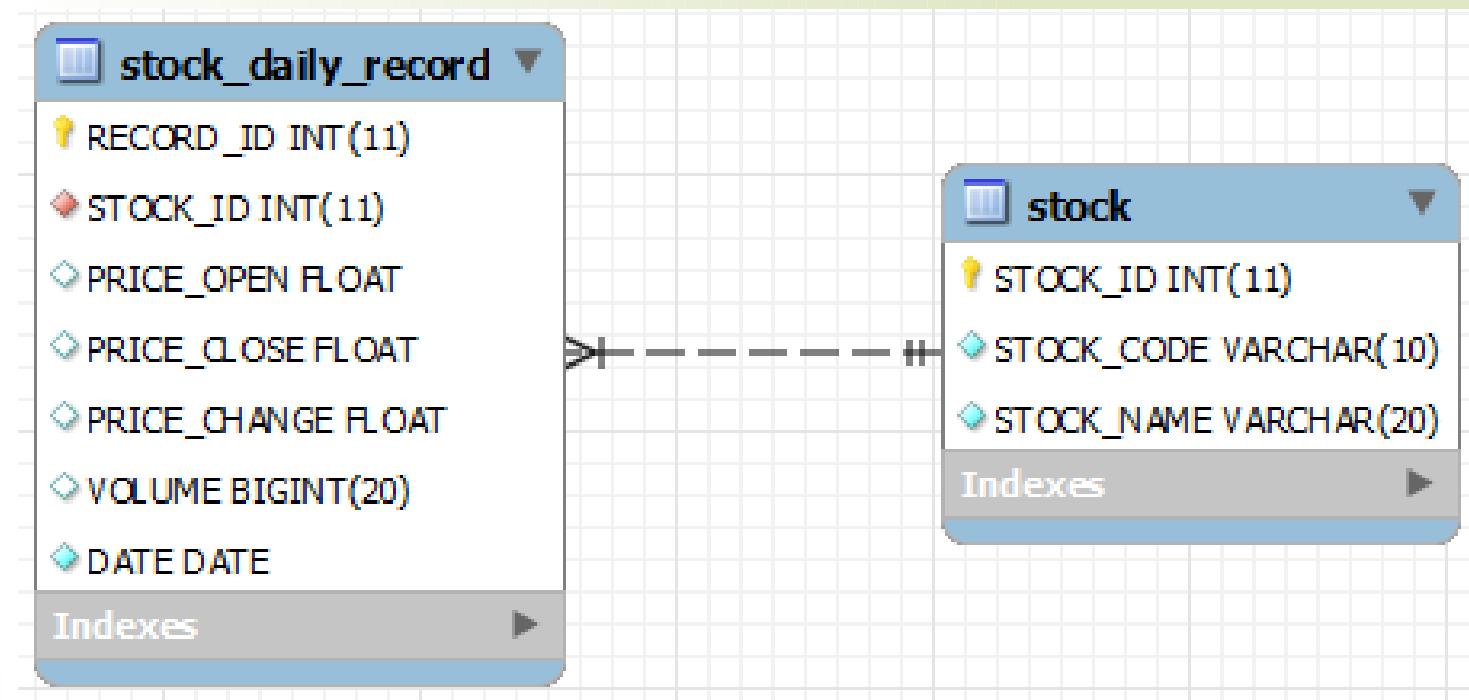
## Ví dụ transaction trong hibernate

- ▶ Ứng dụng mô phỏng việc chuyển tiền trong ngân hàng.

## VI. Chiến lược Fetching trong Hibernate

- ▶ Hibernate có các chiến lược fetching tối ưu hóa việc tạo ra các câu lệnh select để lấy dữ liệu.
- ▶ Chiến lược fetching được khai báo trong mapping relationship để định nghĩa cách Hibernate lấy các Collections và entities liên quan.
- ▶ **fetch-“join”**: tắt lazy loading, luôn luôn lấy tất cả collections và entities
- ▶ **fetch-“select” (default)** : Lazy load tất cả collections và entities
- ▶ **batch-size=”N”** : Lấy ‘N’ collections hoặc entities, \*Not record\*.
- ▶ **fetch-“subselect”** : Nhóm các collection của nó vào trong các câu lệnh select con

# Example



# Fetch="join"

```
//call select from stock and stock_daily_record  
Stock stock = (Stock)session.get(Stock.class, 1);  
Set sets = stock.getStockDailyRecords();  
  
//no extra select  
for ( Iterator iter = sets.iterator();iter.hasNext(); ) {  
    StockDailyRecord sdr = (StockDailyRecord) iter.next();  
    System.out.println(sdr.getRecordId() + " " + sdr.getDate());  
}
```

```
Hibernate:  
select ...  
from  
| mkyong.stock stock0_  
left outer join  
| mkyong.stock_daily_record stockdaily1_  
| on stock0_.STOCK_ID=stockdaily1_.STOCK_ID  
where  
| stock0_.STOCK_ID=?
```

Hibernate đã tạo ra chỉ một câu lệnh select, nó lấy tất cả collection liên quan khi Stock được khởi tạo  
**session.get(Stock.class, 1)**

# Fetch="select" (default)

```
//call select from stock and stock_daily_record  
Stock stock = (Stock)session.get(Stock.class, 1);  
Set sets = stock.getStockDailyRecords();  
  
//no extra select  
for ( Iterator iter = sets.iterator();iter.hasNext(); ) {  
    StockDailyRecord sdr = (StockDailyRecord) iter.next();  
    System.out.println(sdr.getRecordId() + " " + sdr.getDate());  
}
```

Hibernate:

```
select ...from mkyong.stock  
where stock0_.STOCK_ID=?
```

Hibernate:

```
select ...from mkyong.stock_daily_record  
where stockdaily0_.STOCK_ID=?
```

► Hibernate tạo ra 2 câu select

- Câu select đầu tiên được gọi khi thực hiện – **session.get(Stock.class, 1)**
- Câu thứ 2 khi thực hiện lệnh – **sets.iterator()** lần đầu tiên.

# batch-size="N"

```
List<Stock> list = session.createQuery("from Stock").list();
for(Stock stock : list){
    Set sets = stock.getStockDailyRecords();
    for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
        StockDailyRecord sdr = (StockDailyRecord) iter.next();
        System.out.println(sdr.getDailyRecordId());
        System.out.println(sdr.getDate());
    }
}
```

- ▶ Nó định nghĩa bao nhiêu collections được lấy một lần.
- ▶ Ex : bảng Stock có 20 dòng. Với batch-size="10", Hibernate sẽ tạo ra 3 câu lệnh select:
  - ▶ Câu 1 : khi gọi **List<Stock> list = session.createQuery("from Stock").list();**
  - ▶ Câu 2 : lấy 10 Collection<StockDailyRecord> liên quan với 10 Stock đầu tiên.
  - ▶ Câu 3 : lấy 10 Collection<StockDailyRecord> liên quan với 10 Stock còn lại

```
Hibernate:
select ...
from mkyong.stock stock0_
where
stockdaily0_.STOCK_ID in (
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
)
```

```
Hibernate:
select ...
from mkyong.stock_daily_record stockdaily0_
where
stockdaily0_.STOCK_ID in (
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
)
```

# Fetch="subselect"

```
List<Stock> list = session.createQuery("from Stock").list();
for(Stock stock : list){
    Set sets = stock.getStockDailyRecords();
    for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
        StockDailyRecord sdr = (StockDailyRecord) iter.next();
        System.out.println(sdr.getDailyRecordId());
        System.out.println(sdr.getDate());
    }
}
```

```
Hibernate:
select ...
from mkyong.stock stock0_
where
stockdaily0_.STOCK_ID in (
    select
        stock0_.STOCK_ID
    from
        mkyong.stock_daily_record stockdaily0_
)
```

- Hibernate tạo ra 2 câu select
- Một câu select lấy tất cả Stock
- Câu thứ 2 lấy tất cả collection liên quan trong subquery.

## VII. HQL (Hibernate Query Language)

- ▶ Hibernate Query Language (HQL) là một ngôn ngữ truy vấn hướng đối tượng, tương tự như SQL, nhưng thay vì hoạt động trên bảng, cột, HQL làm việc với các đối tượng persistent và các thuộc tính của nó. HQL sẽ chuyển sang câu lệnh SQL tùy thuộc vào CSDL đang sử dụng
- ▶ Có thể truy vấn sử dụng các interface hoặc superclass
- ▶ Nên sử dụng HQL để tránh rắc rối trong việc thay đổi CSDL, ưu điểm của tính năng tạo chuyển đổi SQL của Hibernate

# Truy vấn với Inheritance

- ▶ Query : *from Animal as animal* trả về tất các đối tượng của các lớp hiện thực *Animal* (interface) hoặc các lớp kế thừa *Animal* (class)

# Mệnh đề FROM , WHERE

- ▶ Sử dụng **FROM** lấy các đối tượng tương ứng với các dòng trong CSDL

- ▶ Ví dụ :

```
String hql = "FROM Employee";  
Query query = session.createQuery(hql);  
List results = query.list();
```

- ▶ Sử dụng **WHERE** lấy các đối tượng với theo điều kiện

- ▶ Ví dụ :

```
String hql = "FROM Employee E WHERE E.id = 10";  
Query query = session.createQuery(hql);  
List results = query.list();
```

# Mệnh đề SELECT, ORDER BY

- Khi chỉ cần lấy các đối tượng chỉ chứa một số thuộc tính sử dụng **SELECT**

```
String hql = "SELECT E.firstName FROM Employee E";
```

```
Query query = session.createQuery(hql);
```

```
List results = query.list();
```

- Khi cần sắp xếp các đối tượng theo thuộc tính, sử dụng **ORDER BY**

```
String hql = "FROM Employee E WHERE E.id > 10 " +
```

```
"ORDER BY E.firstName DESC, E.salary DESC ";
```

```
Query query = session.createQuery(hql);
```

```
List results = query.list();
```

# HQL hỗ trợ các câu lệnh như SQL

- ▶ Select, Update, delete
- ▶ Joins (inner join, left outer join, right outer join, full join)
- ▶ Aggregate functions (avg(), sum(), min(), max(), count(\*), ...)
- ▶ Where , group by, order by
- ▶ subqueries

# Sử dụng tham số trong HSQL

- ▶ HQL hỗ trợ sử dụng tham số

- ▶ Ví dụ :

```
String hql = "FROM Employee E WHERE E.id =  
:employee_id";
```

```
Query query = session.createQuery(hql);
```

```
query.setParameter("employee_id",10);
```

```
List results = query.list();
```



Cảm ơn các bạn đã theo dõi !