**Bài 6: Bean-Managed Persistent Entity Bean (BMP)**

**I. Khái niệm**

1. Bean-Managed Persistence

- Với BMP, người viết bean phải viết tường minh các triệu gọi dùng để truy xuất cơ sở dữ liệu. Truy xuất dữ liệu có thể được mã hóa bên trong bean hoặc đóng gói trong đối tượng truy xuất dữ liệu (DAO – Data Access Object) như một phần của bean.

# EJB Container

JDBC Driver

JDBC/SQLJ

SQL

Database

- BMP được chọn dùng khi:

* Muốn điều khiển hoàn toàn việc truy xuất cơ sở dữ liệu, như viết các truy vấn tối ưu, select với yêu cầu phức tạp.
* Muốn viết thao tác truy xuất cho các hệ quản trị cơ sở dữ liệu cũ (legacy system), ERP…
* Nơi lưu trữ không phải là một hệ quản trị cơ sở dữ liệu.

Tuy nhiên, mối liên hệ quá chặt chẽ giữa BMP và cơ sở dữ liệu nó thao tác trên đó là một yếu điểm của BMP.

- BMP cũng như các Entity Bean khác, có 3 trạng thái:

* Does Not Exits: không tồn tại trong bộ nhớ.
* Pooled: có mặt trong instance pool nhưng chưa sẵn sàng hoạt động.
* Ready: sẵn sàng đáp ứng các yêu cầu từ client cho EJB Object ủy nhiệm đến.

*Class.newInstance()*

*setEntityContext()*

System Exception

EJB Container



Client

**DB**



**DB**

*ejbStore()*

*ejbHome<Name>(...)*

*ejbFind<Name>(...)*

*ejbCreate(...)*

*ejbPostCreate(...)*

*ejbActivate()*

*ejbLoad()*

*<method>(...)*

*<method>(...)*

*<method>(...)*

*Object.finalize()*

*unsetEntityContext()*

*ejbRemove()*

*ejbPassivate()*

**Activate** ejbActivate() → ejbLoad()

**Passivate** ejbStore() → ejbPassivate()

2. Vòng đời của Entity Bean

- Vòng đời của bean bắt đầu khi container gọi **Class.newInstance()**, sau đó liên kết bean với đối tượng **EntityContext** chứa thông tin môi trường bằng cách gọi phương thức **setEntityContext()**, bean hình thành và được đặt vào instance pool và ở trạng thái Pooled. Bean trong pool không liên kết với bất kỳ cơ sở dữ liệu và một EJB Object nào, container có thể dùng bean này để thực hiện bất kỳ một phương thức home hoặc finder nào.

- Khi container chọn bean để phục vụ một triệu gọi của client đến EJB Object, bean chuyển sang trạng thái Ready, liên kết với cơ sở dữ liệu và với EJB Object cụ thể. Có hai cách chuyển đến trạng thái Ready:

* Client gọi phương thức **create()** của Home Object và như vậy container sẽ gọi lần lượt các phương thức **ejbCreate()** và **ejbPostCreate()**.
* Container gọi phương thức **ejbActivate()** để kích hoạt lại các bean đang thụ động, cấp phát tài nguyên như socket cho bean..

- Trong trạng thái Ready, bean liên kết với một EJB Object. Container gọi các business method trên bean tùy theo các method mà EJB Object ủy nhiệm đến. Container cũng đồng bộ trạng thái của bean với cơ sở dữ liệu bằng cách dùng các phương thức **ejbLoad()** và **ejbStore()**.

- Container có thể chuyển bean trở lại trạng thái Pooled, điều này xảy ra khi:

* Client gọi phương thức **remove()** và như vậy container sẽ gọi phương thức **ejbRemove()**.
* Container gọi phương thức **ejbPassivate()** để chuyển bean sang trạng thái thụ động.

- Cuối vòng đời của bean, container sẽ loại bean ra khỏi pool và gọi phương thức **unsetEntityContext()** tách bean ra khỏi đối tượng **EntityContext** chứa thông tin môi trường liên quan đến nó.

3. Chi tiết hoạt động của BMP

Client

Container

Bean Instance

Database

setEntityContext()

Thêm một thực thể mới đến Pool

Phục vụ một phương thức finder

Phục vụ phương thức create

beanClass.newInsance()

Phục vụ một business method

trả về một EJB object hay một collection EJB object

bussinessMethod(args)

ejbObj.businessMethod(args)

ejbStore()

ejbPassivate()

ejbObj.remove()

hoặc

home.remove()

Phục vụ phương thức remove

home.find<METHOD>(args)

ejbFind<METHOD>(args)

tìm trong cơ sở dữ liệu

trên cơ sở args

trả về một PK hay

một collection PK

trả về EJB object mới

home.create(args)

ejbCreate(args)

chèn một hàng

vào cơ sở dữ liệu

trả về một PK

ejbPostCreate(args)

ejbObj. businessMethod(args)

hoặc

ejbHome. businessMethod(args)

cập nhật hàng thể hiện

vào cơ sở dữ liệu

Passivating một thực thể bean

ejbActivate()

đọc dữ liệu vào thực thể từ cơ sở dữ liệu

ejbLoad()

bussinessMethod(args)

Activating một thực thể bean

xóa hàng thể hiện

trong cơ sở dữ liệu

ejbRemove()

unsetEntityContext()

Loại một thực thể khỏi Pool

4. Phương thức callback, Finder và phương thức Home

a) Callback

- Phương thức **ejbCreate()** được gọi khi ghi trạng thái của bean vào cơ sở dữ liệu, bean liên kết với đại diện EJB Object của nó. Như vậy một hàng mới được tạo và được ghi vào cơ sở dữ liệu. Đặc tả EJB từ 2.0 trở đi cho phép nạp chồng phương thức này dưới dạng **ejbCreate<SUFFIX>()**, cần có phương thức **create<SUFFIX>()** tương ứng trong home interface. Truy vấn SQL tương ứng là **INSERT**. Phương thức **create()** là tùy chọn trong entity bean.

- Phương thức **ejbPostCreate()** được gọi sau khi gọi **ejbCreate()**. Phương thức này dùng cho một số tác vụ như: giải quyết các vấn đề phát sinh khi chèn hàng vào cơ sở dữ liệu, thiết lập mối quan hệ giữa các bean có liên quan với nhau sau khi chúng hình thành, chuyển tham chiếu của bean đến một bean khác, thay đổi các thông số transaction, ...

- Phương thức **ejbRemove()** dùng để tách mối liên kết giữa bean với dữ liệu tương ứng trong cơ sở dữ liệu. Truy vấn SQL tương ứng là **DELETE**.

- Phương thức **ejbLoad()** dùng để nạp dữ liệu từ cơ sở dữ liệu vào bean tương ứng (còn gọi là refresh dữ liệu), đồng bộ trạng thái của bean với hàng dữ liệu mà nó ánh xạ. Phương thức này được gọi trước khi các business method được thực hiện. Truy vấn SQL tương ứng là **SELECT \***.

- Phương thức **ejbStore()** dùng để ghi trạng thái (dữ liệu của lớp bean) vào cơ sở dữ liệu, đồng bộ trạng thái của bean với hàng dữ liệu mà nó ánh xạ. Phương thức này được gọi sau khi business method thực hiện xong. Truy vấn SQL tương ứng là **UPDATE**.

- **ejbLoad()** và **ejbStore()** là các phương thức tự động đồng bộ Entity Bean với hàng dữ liệu mà nó ánh xạ trong cơ sở dữ liệu. Nghĩa là khi thay đổi Entity Bean, **ejbStore()** sẽ được gọi để cập nhật cơ sở dữ liệu; khi thay đổi cơ sở dữ liệu, **ejbLoad()** sẽ được gọi để cập nhật Entity Bean.

- Container sẽ gọi phương thức **ejbStore()** trước khi gọi phương thức **ejbPassivate()**, giải phóng bean. Sau khi container gọi phương thức **ejbActivate()**, phương thức **ejbLoad()** sẽ được gọi để nạp lại dữ liệu cho bean từ cơ sở dữ liệu.

- Phương thức **setEntityContext()** dùng để liên kết bean với đối tượng **javax.ejb.EntityContext**. Thông qua đối tượng này EJB nhận những thông tin liên quan đến môi trường.

- Phương thức **unsetEntityContext()** tách bean khỏi môi trường, giải phóng tài nguyên do **setEntityContext()** cấp, thực thể bean sẵn sàng để hủy.

b) Finder

- Các phương thức Finder là cần thiết do thực tế thường có khuynh hướng làm việc với một *hàng dữ liệu có sẵn*, không phải là hàng dữ liệu vừa tạo mới. Mỗi phương thức có tên **find<SUFFIX>()** trong home interface, phải có một phương thức **ejbFind<SUFFIX>()** tương ứng với cùng đối số trong lớp bean. Khi một phương thức **find()** được triệu gọi trên Home Object, container sẽ ủy nhiệm phương thức **ejbFind()** tương ứng trên bean thực hiện yêu cầu. Truy vấn SQL tương ứng là **SELECT PrimaryKey WHERE**.

- Kết quả trả về của các phương thức **find<SUFFIX>()** là một đối tượng kiểu EJB Object hoặc một **Collection** các đối tượng kiểu EJB Object, trong lúc kết quả trả về của các phương thức **ejbFind<SUFFIX>()** là khóa chính hoặc một **Collection** các khóa chính.

- Phải có ít nhất một Finder: **findByPrimaryKey( PrimaryKey )**.

c) Phương thức Home

- Đôi khi chúng ta cần các phương thức trên một Entity bean mà không chỉ định bất kỳ thực thể (hàng dữ liệu) cần lấy nào. Ví dụ, đếm số account trong một bảng. Các phương thức Home dùng thực hiện các tác vụ này.

- Các phương thức home là các business method đặc biệt bởi vì chúng được gọi từ bean trong pool, trước khi bean liên kết với dữ liệu chỉ định nào đó. Client gọi các phương thức Home thông qua home interface (local hoặc remote). Mỗi phương thức có tên **x<SUFFIX>()** trong home interface, phải có một phương thức **ejbHomeX<SUFFIX>()** tương ứng với cùng đối số trong lớp bean.

**Container**

**JDBC Driver**

**Database**

**JNDI**

**Service**

**Client**

3. create

2. lookup đến home trong JNDI

5. remove

JDBC/SQLJ

4. Gọi các business method

Các phương thức callback

SQL

4.1 Gọi các business method

3.1 Tạo bean

5.1 ejbRemove

ejbLoad

ejbStore

ejbPassivate

ejbActivate

1. Đăng ký EJB

1. Container đăng ký tất cả các bean đã được triển khai với JNDI.

2. Client tìm thấy (**lookup()**) home interface của bean thông qua JNDI.

3. Client dùng phương thức **create()** của home interface để tạo ra EJB Object. Khi client triệu gọi phương thức **create()** của home interface, container triệu gọi phương thức **ejbCreate()** tương ứng của bean.

4. Client triệu gọi các business method trên EJB Object, container ủy nhiệm các lời gọi này đến bean.

5. Client gọi phương thức **remove()** của EJB Object, container gọi **ejbRemove()** của bean tương ứng.

Container đồng bộ trạng thái của bean với cơ sở dữ liệu bằng cách dùng các phương thức **ejbLoad()** và **ejbStore()**.

5. Lớp Primary Key (khóa chính)

- Primary Key là lớp đối tượng dùng để chỉ định duy nhất một entity bean. Primary Key có thể là kiểu serializable bất kỳ nào, bao gồm các kiểu có sẵn (**Integer**, **Double**, **String**, ...) hoặc các lớp được định nghĩa bởi người phát triển bean.

- Có hai kiểu Primary Key:

* Single-Field: Primary Key ánh xạ với một field định nghĩa trong lớp bean.
* Compound: lớp Primary Key (cài đặt giao diện **java.io.Serializable**) do người dùng định nghĩa ánh xạ đến nhiều field trong lớp bean.

- Chú ý khi tạo lớp Primary Key sao cho hợp lệ trong RMI/IIOP:

* Phải cài đặt giao diện **java.io.Serializable**.
* Tất cả các field phải **public**.
* Phải định nghĩa cùng lúc các phương thức **equals()** và **hashCode()** để thao tác dễ dàng trên Primary Key trong **Collection**.
* Không cài đặt giao diện **java.rmi.Remote**.

- Deployment Descriptor cũng phải định nghĩa lớp Primary Key cần cho triển khai:

**<prim-key-class>cShipPK</prim-key-class>**

6. Lớp và giao diện

a) Client truy xuất từ xa (remote client view)

Identity mamagement

Home method

Appilcation Logic

PK mamagement

State management

interface EJBHome

+ EJBMetaData getEJBMetaData( )  
+ HomeHandle getHomeHandle( )

+ void remove( Handle )

throws RemoveException

+ void remove(<PK> )

throws RemoveException

interface <name>Home

+ <remote> create( <args> )

throws CreateException

+ <remote> findByPrimaryKey( <PK> )

throws FinderException

+ <remote coll>

find<methode>( <args> )

throws FinderException

+ <type> <name>( <args> )

throws SomeException

interface EJBObject

+ void remove( Handle )

throws RemoveException

+ EJBHome getEJBHome( )

+ Handle getHandle( )

+ Object getPrimaryKey( )

+ boolean isIdentical( EJBObject )

interface <name>

+ <ret type> business\_1( <args> )

throws SomeException

+ <ret type> business\_2( <args> )

throws SomeException

+ abstract <*name*>Bean

+ <type> AttributeKey

+ <type> AttributeX

+ <type> AttributeY

+ void ejbLoad( )

+ void ejbStore( )

+ void ejbActivate( )

+ void ejbPassivate( )

+ void setEntitynContext( <arg> )

+ void unsetEntityContext( )

+ void ejbRemove( )

throws RemoveException

+ <PK> ejbCreate( <args> )

throws CreateException

+ <PK> ejbFindByPrimaryKey( <PK> )

throws FinderException

+ <PK collection>

ejbFind<methode>( <args> )

throws FinderException

+ <type>ejbHome<Name>( <arg> )

throws SomeException

+ <ret type> business\_1( <args> )

throws SomeException

+ <ret type> business\_2( <args> )

throws SomeException

interface EntityBean

↑ throws RemoteException ↑ ↑ throws EJBException ↑

+ <name>PK

<type> <attributeKey>

interface java.io.Serializable

b) Client truy xuất cục bộ (local client view)

Identity mamagement

Home method

Appilcation Logic

PK mamagement

State management

interface EJBLocalHome

+ void remove(<PK> )

throws RemoveException

interface <name>Home

+ <local> create( <args> )

throws CreateException

+ <local> findByPrimaryKey( <PK> )

throws FinderException

+ <local collection>

find<method>( <args> )

throws FinderException

+ <type> <name>( <args> )

throws SomeException

interface EJBLocalObject

+ void remove( Handle )

throws RemoveException

+ EJBLocalHome getEJBLocalHome( )

+ Object getPrimaryKey( )

+ boolean isIdentical( EJBLocalObject )

interface <name>

+ <ret type> business\_1( <args> )

throws SomeException

+ <ret type> business\_2( <args> )

throws SomeException

+ abstract <*name*>Bean

+ <type> AttributeKey

+ <type> AttributeX

+ <type> AttributeY

+ <type> AttributeZ

+ void ejbLoad( )

+ void ejbStore( )

+ void ejbActivate( )

+ void ejbPassivate( )

+ void setEntityContext( <arg> )

+ void unsetEntityContext( )

+ void ejbRemove( )

throws RemoveException

+ <PK> ejbCreate( <args> )

throws CreateException

+ <PK> ejbFindByPrimaryKey( <PK> )

throws FinderException

+ <PK collection>

ejbFind<methode>( <args> )

throws FinderException

+ <type>ejbHome<Name>( <arg> )

throws SomeException

+ <ret type> business\_1( <args> )

throws SomeException

+ <ret type> business\_2( <args> )

throws SomeException

interface EntityBean

↑ throws EJBException ↑

+ <name>PK

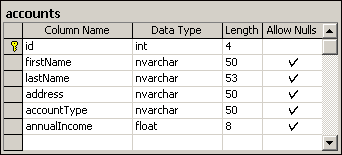
<type> <attributeKey>

interface java.io.Serializable

**II. Thiết kế**

**A. Tạo BMP**

- Cơ sở dữ liệu cho ứng dụng: **Bank**, bảng **accounts**.



- Entity Bean là ánh xạ của một hàng trong bảng dữ liệu, khi xây dựng bean ta dựa theo phương pháp O-R mapping và bảng dữ liệu mà EJB ánh xạ để hình thành lớp Bean.

1. Lớp Primary Key (khóa chính)

- Primary Key định nghĩa thuộc tính dùng để định vị một hàng dữ liệu (tức một bean) trong cơ sở dữ liệu. Thường chỉ cần một thuộc tính, nhưng đôi khi cần một bộ thuộc tính mới xác định được một hàng duy nhất trong bảng. Nếu Primary Key chỉ là kiểu serializable có sẵn thì không cần phải viết lớp Primary Key.

**package myejb.entity.bmp;**

**public class AccountPK implements java.io.Serializable {**

**private int id;**

**public AccountPK() { }**

**public AccountPK( int id ) {**

**this.id = id;**

**}**

**public void setId( int id ) {**

**this.id = id;**

**}**

**public int getId() {**

**return id;**

**}**

**public String toString() {**

**return String.valueOf( id );**

**}**

**public int hashCode() { return id; }**

**public boolean equals( Object obj ) {**

**if ( obj == null || !( obj instanceof AccountPK ) ) return false;**

**AccountPK other = ( AccountPK )obj;**

**return ( this.id == other.id );**

**}**

**}**

2. Tạo lớp EJB

- Lớp EJB cho BMP viết phức tạp vì người cung cấp bean phải chịu trách nhiệm thực hiện các thao tác truy xuất cơ sở dữ liệu. Lớp Bean gồm có:

* Dữ liệu riêng ánh xạ các field của bảng dữ liệu
* Các phương thức tác động lên các field dữ liệu trên, chính là các business logic của bean.
* Các phương thức hỗ trợ (helper) thường là các phương thức truy xuất dữ liệu, gọi là data access logic.
* Các phương thức callback gọi bởi Container, các Finder, các phương thức Home.

a) Dữ liệu riêng của bean và các business logic

- Bean phải có dữ liệu riêng, chính là thể hiện các field của hàng dữ liệu mà bean ánh xạ bên trong nó.

**private int id;**

**private String firstName;**

**private String lastName;**

**private String address;**

**private String accountType;**

**private double annualIncome;**

- Dữ liệu riêng này được truy xuất (vào/ra) bởi các getter/setter (gọi chung là các accessor). Đây cũng là các business method sẽ được client triệu gọi. Tiếp đầu ngữ set/get là quy ước và số lượng các accessor là tùy yêu cầu, không phải bắt buộc.

**//** Business methods (Getter / Setter)

**public String getFirstName() {**

**return firstName;**

**}**

**public String getLastName() {**

**return lastName;**

**}**

**public String getAddress() {**

**return address;**

**}**

**public String getAccountType() {**

**return accountType;**

**}**

**public double getAnnualIncome() {**

**return annualIncome;**

**}**

**BMP**

Client

Tier

Bussiness

Tier

EIS

Tier

Database

BMP Fields

Client

*Network*

Nhiều triệu gọi qua mạng và code bị trộn lẫn

- Cách cài đặt trên là kinh điển. Nhưng thực tế để giảm thiểu số lời gọi các business method của EJB qua mạng, người ta thường dùng design pattern Transfer Object, còn gọi là Value Object. Một Transfer Object là một đối tượng POJO[[1]](#footnote-2) gọn nhẹ, có khả năng serialize (nghĩa là cài đặt giao diện **java.io.Serializable**), nhóm các field dữ liệu thành một cấu trúc logic đơn, và vì vậy chỉ phục vụ cho một lời gọi duy nhất.

- Thay vì phải gọi nhiều getter/setter, ta gom các field dữ liệu vào đối tượng Transfer Object, rồi chỉ gọi getter/setter cho đối tượng này. Lớp mô tả Transfer Object:

**package myejb.entity.bmp;**

**public class AccountTO implements java.io.Serializable {**

**private int id;**

**private String firstName;**

**private String lastName;**

**private String address;**

**private String accountType;**

**private double annualIncome;**

**private AccountPK primaryKey;**

**public AccountTO() {**

**primaryKey = new AccountPK();**

**}**

**public AccountTO( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) {**

**primaryKey = new AccountPK( id );**

**setId( id );**

**Client**

**Component**

<<POJO>>

**Transfer Object**

accesses

1..\*

1..\*

creates/uses

creates/uses

**setFirstName( firstName );**

**setLastName( lastName );**

**setAddress( address );**

**setAccountType( accountType );**

**setAnnualIncome( annualIncome );**

**}**

**public AccountPK getPrimaryKey()**

**{ return primaryKey; }**

**public void setPrimaryKey( AccountPK primaryKey ) {**

**this.primaryKey = primaryKey;**

**setId( primaryKey.getId() );**

**}**

**public int getId()**

**{ return this.id; }**

**public void setId( int id ) {**

**this.id = id;**

**primaryKey.setId( id );**

**}**

**public String getFirstName()**

**{ return this.firstName; }**

**public void setFirstName( String firstName )**

**{ this.firstName = firstName; }**

**public String getLastName()**

**{ return this.lastName; }**

**public void setLastName( String lastName )**

**{ this.lastName = lastName; }**

**public String getAddress()**

**{ return this.address; }**

**public void setAddress( String address )**

**{ this.address = address; }**

**public String getAccountType()**

**{ return this.accountType; }**

**public void setAccountType( String accountType )**

**{ this.accountType = accountType; }**

**public double getAnnualIncome()**

**{ return this.annualIncome; }**

**public void setAnnualIncome( double annualIncome )**

**{ this.annualIncome = annualIncome; }**

**}**

- Business method của BMP khi dùng Transfer Object:

**//** Business methods (Getter / Setter)

**public AccountTO getAccountTO() {**

**return new AccountTO( id, firstName, lastName, address, accountType, annualIncome );**

**}**

**public void setAccountTO( AccountTO updateObj ) {**

**this.id = updateObj.getId();**

**this.firstName = updateObj.getFirstName();**

**this.lastName = updateObj.getLastName();**

**this.address = updateObj.getAddress();**

**this.accountType = updateObj.getAccountType();**

**this.annualIncome = updateObj.getAnnualIncome();**

**}**

b) Phương thức hỗ trợ (helper method)

Database

**BMP**

Client

Tier

Bussiness

Tier

EIS

Tier

*Helper*

Tách Data Access Logic thành các phương thức Helper

Dùng Transfer Object để giảm các triệu gọi qua mạng

Client

BMP Fields

*Network*

**Refactoring 1**

- Ngoài business logic, trong BMP còn có data access logic, chính là các thao tác truy xuất cơ sở dữ liệu (CRUD[[2]](#footnote-3)) được cài đặt trong các phương thức callback, các finder, các phương thức ejbHome. Để dễ cài đặt, nên tách biệt phần data access logic này thành các phương thức hỗ trợ (, còn gọi là utility method). Các phương thức helper chỉ dùng bên trong lớp nên phạm vi truy xuất thường là **private**, khi Container gọi các phương thức callback, các phương thức callback sẽ gọi các phương thức helper này.

- Các phương thức hỗ trợ bao gồm:

* Tạo kết nối cơ sở dữ liệu: kết nối cơ sở dữ liệu thường được thực hiện thông qua một driver JDBC Type1 (cầu nối JDBC-ODBC) hoặc Type4 (Driver Java). Nguồn dữ liệu (đối tượng **DataSource**) được lookup nhờ JNDI rồi sử dụng để trả về đối tượng kết nối. Đây là đặc điểm kết nối cơ sở dữ liệu của gói JDBC 2.0 Optional.

**private Connection getConnection() throws SQLException {**

**DataSource ds = null;**

**try {**

**Context ctx = new InitialContext();**

**ds = ( DataSource )ctx.lookup( "java:/Bank" );**

**} catch ( Exception e ) {**

**System.out.println( "Couldn't get database" );**

**e.printStackTrace();**

**}**

**return ds.getConnection();**

**}**

Cũng có thể kết nối trực tiếp với nguồn dữ liệu đăng ký trong DSN. Cách này không thể hiện điểm mạnh của hệ thống phân tán và thực tế không dùng đến. Với các này không cần đăng ký **<resource-ref>** trong các DD như **ejb-jar.xml**.

**private Connection getConnection() throws Exception {**

**Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );**

**return DriverManager.getConnection( "jdbc:odbc:Bank", "sa", "" );**

**}**

* Các phương thức dùng để chèn hoặc xóa một hàng dữ liệu của bảng, sẽ được dùng cho các phương thức **ejbCreate()** (chèn, truy vấn SQL **INSERT**) và **ejbRemove()** (xóa, truy vấn SQL **DELETE**) của bean.

**private void insertRow( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws SQLException {**

**Connection conn = this.getConnection();**

**PreparedStatement pstmt = conn.prepareStatement(**

**"INSERT INTO accounts VALUES ( ?, ?, ?, ?, ?, ? )" );**

**pstmt.setInt( 1, id );**

**pstmt.setString( 2, firstName );**

**pstmt.setString( 3, lastName );**

**pstmt.setString( 4, address );**

**pstmt.setString( 5, accountType );**

**pstmt.setDouble( 6, annualIncome );**

**pstmt.executeUpdate();**

**conn.close();**

**}**

**private void removeRow( AccountPK pk ) throws SQLException, RemoveException {**

**Connection conn = this.getConnection();**

**PreparedStatement pstmt = conn.prepareStatement( "DELETE FROM Accounts WHERE id = ?" );**

**pstmt.setInt( 1, pk.getId() );**

**boolean result = ( pstmt.executeUpdate() == 0 );**

**conn.close();**

**if ( result ) throw new RemoveException( "Account " + pk.getId() +**

**" failed to be removed from the database" );**

**}**

* Các phương thức dùng để nạp hoặc lưu một hàng dữ liệu của bảng, sẽ được dùng cho các phương thức **ejbLoad()** (truy vấn SQL **SELECT \***) và **ejbStore()** (truy vấn SQL **UPDATE**) của bean.

**private void loadRow( AccountPK pk ) throws SQLException {**

**Connection conn = this.getConnection();**

**PreparedStatement pstmt = conn.prepareStatement(**

**"SELECT \* FROM accounts WHERE id = ?" );**

**pstmt.setInt( 1, pk.getId() );**

**ResultSet rs = pstmt.executeQuery();**

**rs.next();**

**this.firstName = rs.getString( 2 );**

**this.lastName = rs.getString( 3 );**

**this.address = rs.getString( 4 );**

**this.accountType = rs.getString( 5 );**

**this.annualIncome = rs.getDouble( 6 );**

**rs.close();**

**conn.close();**

**}**

**private void storeRow( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws SQLException {**

**Connection conn = this.getConnection();**

**PreparedStatement pstmt = conn.prepareStatement(**

**"UPDATE accounts SET firstName = ?, lastName = ?, address = ?,"**

**+ " accountType = ?, annualIncome = ? WHERE id = ?" );**

**pstmt.setString( 1, firstName );**

**pstmt.setString( 2, lastName );**

**pstmt.setString( 3, address );**

**pstmt.setString( 4, accountType );**

**pstmt.setDouble( 5, annualIncome );**

**pstmt.setInt( 6, id );**

**pstmt.executeUpdate();**

**conn.close();**

**}**

* Các phương thức dùng để tìm kiếm một hoặc nhiều hàng dữ liệu của bảng, sẽ được dùng cho các phương thức Finder (truy vấn SQL **SELECT WHERE**) của bean.

**private boolean findByPK( AccountPK pk ) throws SQLException {**

**boolean result = false;**

**Connection conn = this.getConnection();**

**PreparedStatement pstmt = conn.prepareStatement(**

**"SELECT id FROM accounts WHERE id = ?" );**

**pstmt.setInt( 1, pk.getId() );**

**ResultSet rs = pstmt.executeQuery();**

**result = rs.next();**

**rs.close();**

**conn.close();**

**return result;**

**}**

**private Collection findByFName( String firstName ) throws SQLException {**

**Vector accounts = new Vector();**

**Connection conn = this.getConnection();**

**PreparedStatement pstmt = conn.prepareStatement(**

**"SELECT id FROM accounts WHERE firstName = ?" );**

**pstmt.setString( 1, firstName );**

**ResultSet rs = pstmt.executeQuery();**

**while ( rs.next() ) {**

**int id = rs.getInt( 1 );**

**accounts.addElement( new AccountPK( id ) );**

**}**

**rs.close();**

**conn.close();**

**return accounts;**

**}**

* Các phương thức dùng để truy xuất các thuộc tính riêng biệt của một hoặc nhiều hàng dữ liệu của bảng, sẽ được dùng cho các phương thức **ejbHome<…>()** của bean.

**private int selectTopID() throws SQLException {**

**int n = 99; //** The first ID is 100, for empty database

**Connection conn = this.getConnection();**

**Statement stmt = conn.createStatement();**

**ResultSet rs = stmt.executeQuery( "SELECT TOP 1 id FROM accounts ORDER BY id DESC" );**

**if ( rs.next() ) n = rs.getInt( 1 );**

**rs.close();**

**conn.close();**

**return n;**

**}**

c) Các phương thức callback

- Mỗi phương thức calback gọi phương thức helper tương ứng để thực hiện các thao tác truy xuất cơ sở dữ liệu:

* Phương thức **ejbCreate()** dùng tạo một hàng dữ liệu mới nên có các đối số và có thể nạp chồng để tạo ra các truy vấn **INSERT** cần thiết. Ngoài truy vấn **INSERT**, **ejbCreate()** còn khởi tạo cho các field dữ liệu của bean.
* Các phương thức **ejbRemove()** (tương ứng truy vấn **DELETE**), **ejbLoad()** (tương ứng truy vấn **SELECT \***) và **ejbStore()** (tương ứng truy vấn **UPDATE**) tác động lên chính hàng dữ liệu đó nên các tham số cho truy vấn SQL lấy ngay từ các field dữ liệu của bean.

**public AccountPK ejbCreate( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws CreateException {**

**System.out.println( "ejbCreate() called" );**

**try {**

**insertRow( id, firstName, lastName, address, accountType, annualIncome );**

**} catch ( Exception e ) {**

**throw new CreateException( "Failed to create Account " + e.getMessage() );**

**}**

**this.id = id;**

**this.firstName = firstName;**

**this.lastName = lastName;**

**this.address = address;**

**this.accountType = accountType;**

**this.annualIncome = annualIncome;**

**return new AccountPK( id );**

**}**

**public void ejbPostCreate( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) { }**

**public void ejbRemove() throws RemoveException {**

**System.out.println( "ejbRemove() called" );**

**AccountPK pk = ( AccountPK )ctx.getPrimaryKey();**

**try {**

**removeRow( pk );**

**} catch ( Exception e ) {**

**throw new RemoveException( "Failed to remove Account " + e.getMessage() );**

**}**

**}**

**public void setEntityContext( EntityContext ctx ) { this.ctx = ctx; }**

**public void unsetEntityContext() { this.ctx = null; }**

**public void ejbActivate() {**

**System.out.println( "ejbActivate() called" );**

**}**

**public void ejbPassivate() {**

**System.out.println( "ejbPassivate() called" );**

**}**

**public void ejbLoad() {**

**System.out.println( "ejbLoad() called" );**

**AccountPK pk = ( AccountPK )ctx.getPrimaryKey();**

**try {**

**loadRow( pk );**

**} catch ( Exception e ) {**

**throw new EJBException( "ejbLoad: Unable to load from database", e );**

**}**

**}**

**public void ejbStore() {**

**System.out.println( "ejbStore() called" );**

**try {**

**storeRow( id, firstName, lastName, address, accountType, annualIncome );**

**} catch ( Exception e ) {**

**throw new EJBException( "ejbStore: Unable to store into database", e );**

**}**

**}**

- Trong BMP, khi container chuyển bean từ trạng thái Pooled sang trạng thái Ready sẽ không tự động thiết lập khóa chính. Vì vậy, các phương thức **ejbCreate()**cần phải thiết lập khóa chính, đó là lý do cần lưu trữ đối tượng **EntityContext**. Việc lấy lại khóa chính thông qua phương thức **getPrimaryKey()** của đối tượng **EntityContext** có thể thực hiện trong các phương thức **ejbActivate()** hoặc **ejbLoad()** (gọi ngay sau **ejbActivate()**) trong quá trình Activation một bean.

d) Các phương thức Finder

- Các phương thức **ejbFind<…>( <…> )** trả về một đối tượng khóa chính hoặc một Collection các khóa chính, cũng gọi các phương thức helper tương ứng để tiến hành tìm kiếm. Phương thức **ejbFindByPrimaryKey()** xác định khóa chính cần tìm có tồn tại trong cơ sở dữ liệu không, nếu có thì trả về khóa chính cần tìm đó. Phương thức **ejbFind<…>()** khác trả về tập khóa chính của các hàng có thuộc tính cần tìm.

**public AccountPK ejbFindByPrimaryKey( AccountPK key ) throws FinderException {**

**System.out.println( "ejbFindByPrimaryKey() called" );**

**try {**

**boolean result = findByPK( key );**

**if ( !result )**

**throw new ObjectNotFoundException( "Account id " + key.getId() + " not found" );**

**return key;**

**} catch ( Exception e ) {**

**throw new FinderException( "FinderException: " + key.getId() + e.getMessage() );**

**}**

**}**

**public Collection ejbFindByFirstName( String firstName ) throws FinderException {**

**System.out.println( "ejbFindByFirstName() called" );**

**try {**

**Collection c = findByFName( firstName );**

**if ( c.isEmpty() ) throw new ObjectNotFoundException(**

**"Rows for firstname " + firstName + " not found" );**

**return c;**

**} catch ( Exception e ) {**

**throw new FinderException( "FinderException: " + firstName + e.getMessage() );**

**}**

**}**

- Phương thức **ejbFindByPrimaryKey()** đôi khi cài đặt đơn giản hơn, chỉ trả về khóa chính được cấp mà không thực hiện bất cứ truy xuất cơ sở dữ liệu nào:

**public AccountPK ejbFindByPrimaryKey( AccountPK key ) {**

**System.out.println( "ejbFindByPrimaryKey() called" );**

**return key;**

**}**

Cài đặt như trên có lỗi tiềm ẩn do nếu không có lời gọi dữ liệu, cơ sở dữ liệu sẽ không khóa, cho phép các transaction khác thay đổi dữ liệu bên dưới entity bean. Điều này nói chung chấp nhận được nếu finder chỉ đơn giản xác minh sự tồn tại của entity mà không truy xuất một dữ liệu nào khác ngoại trừ khóa chính.

e) Các phương thức ejbHome

- Các phương thức **ejbHome<…>( <…> )** phục vụ cho những tác vụ tổng quát, như đếm số hàng, tính toán trên một tập thuộc tính, xác định một thuộc tính theo yêu cầu, …

**public int ejbHomeGetTopID() throws FinderException {**

**System.out.println( "ejbHomeGetTopID() called" );**

**try {**

**return selectTopID();**

**} catch ( Exception e ) {**

**throw new FinderException( "FinderException: " + e.getMessage() );**

**}**

**}**

3. Các giao diện

- EJB 2.x hỗ trợ cả giao diện truy xuất từ xa và truy xuất cục bộ. Ví dụ minh họa chỉ hiện thực giao diện truy xuất từ xa. Thực tế không nên truy xuất thẳng vào Entity bean mà nên thông qua một bean Session (gọi là bean wrapper) theo design pattern Session façade. Chương sau sẽ thảo luận về vấn đề này.

a) Remote interface

**package myejb.entity.bmp;**

**import javax.ejb.\*;**

**import java.rmi.\*;**

**public interface Account extends EJBObject {**

**public AccountTO getAccountTO() throws RemoteException;**

**public void setAccountTO( AccountTO updateObj ) throws RemoteException;**

**}**

b) Remote Home interface

- Phương thức **create()** và **findByPrimaryKey()** trả về một đối tượng kiểu EJB Object. Các phương thức finder có thể trả về một tập hợp, ví dụ **findByFirstName()** trả về một **Collection** các đối tượng kiểu EJB Object. Phương thức Home có trị trả về tùy mục đích truy xuất.

**package myejb.entity.bmp;**

**import javax.ejb.\*;**

**import java.rmi.\*;**

**import java.util.\*;**

**public interface AccountHome extends EJBHome {**

**public Account create( int id, String firstName, String lastName, String address,**

**String account, double annualIncome ) throws RemoteException, CreateException;**

**public Account findByPrimaryKey( AccountPK key )**

**throws RemoteException, FinderException;**

**public Collection findByFirstName( String name )**

**throws RemoteException, FinderException;**

**public int getTopID() throws RemoteException;**

**}**

4. Deployment Descriptor

- DD **ejb-jar.xml** là tập tin mô tả cho gói **.JAR**:

**<?xml version="1.0"?>**

**<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee**

**http://java.sun.com/xml/ns/j2ee/ejb-jar\_2\_1.xsd">**

**<enterprise-beans>**

**<entity>**

**<ejb-name>Account</ejb-name>**

**<home>myejb.entity.bmp.AccountHome</home>**

**<remote>myejb.entity.bmp.Account</remote>**

**<persistence-type>** cho biết đây là một **BMP**

**<ejb-class>myejb.entity.bmp.AccountBean</ejb-class>**

**<persistence-type>Bean</persistence-type>**

**<prim-key-class>myejb.entity.bmp.AccountPK</prim-key-class>**

**<reentrant>False</reentrant>**

**<resource-ref>**

**<res-ref-name>Bank</res-ref-name>**

**<res-type>javax.sql.DataSource</res-type>**

**ENC**

**<res-auth>Container</res-auth>**

**</resource-ref>**

**</entity>**

**</enterprise-beans>**

**<assembly-descriptor>**

**<container-transaction>**

**<method>**

**<ejb-name>Account</ejb-name>**

**<method-intf>Local</method-intf>**

**<method-name>\*</method-name>**

**</method>**

**<method>**

**<ejb-name>Account</ejb-name>**

**<method-intf>Remote</method-intf>**

**<method-name>\*</method-name>**

**</method>**

**<trans-attribute>Required</trans-attribute>**

**</container-transaction>**

**</assembly-descriptor>**

**</ejb-jar>**

- DD **jboss.xml** là tập tin mô tả của nhà cung cấp (vendor-specific deployment descriptor)

**<?xml version="1.0"?>**

**<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"**

**"http://www.jboss.org/j2ee/dtd/jboss\_4\_0.dtd">**

**<jboss>**

**<enterprise-beans>**

**<entity>**

**<ejb-name>Account</ejb-name>**

**<jndi-name>AccountJNDI</jndi-name>**

**<resource-ref>**

**<res-ref-name>Bank</res-ref-name>**

**<jndi-name>java:/Bank</jndi-name>**

**</resource-ref>**

**</entity>**

**</enterprise-beans>**

**</jboss>**

**B. Tạo client truy xuất BMP**

- Phần client minh họa truy xuất BMP từ xa có hai tab:

* Tab **Registration** tạo một account mới, gọi phương thức **create()** của Remote Home interface. Registration ID của account sẽ đăng ký được tính trước nhờ phương thức Home **getTopID()**.
* Tab **Search** tìm account, gọi các phương thức finder của Remote Home interface: phương thức **findByPrimaryKey()** và phương thức **findByFirstName()**. Dữ liệu kết quả chứa trong Transfer Object được chuyển đến client. Trong bảng kết quả hiển thị, chọn một hàng rồi click phải vào hàng chọn, cho phép xóa hàng dữ liệu đó bằng phương thức **remove( primarykey )** của Remote Home interface (ta không override phương thức này).

- Xem code truy xuất EJB tại các phương thức xử lý sự kiện của Client.

**package clients;**

**import java.util.\*;**

**import javax.swing.\*;**

**import javax.swing.table.\*;**

**import javax.naming.\*;**

**import myejb.entity.bmp.\*;**

**import javax.rmi.\*;**

**import java.awt.event.\*;**

**public class CustomerClient extends JFrame {**

**private JPanel jContentPane = null;**

**private JTabbedPane jTabbedPane = null;**

**private JPanel jPanel = null;**

**private JLabel jLabel = null;**

**private JTextField jTextField = null;**

**private JLabel jLabel1 = null;**

**private JTextField jTextField1 = null;**

**private JLabel jLabel2 = null;**

**private JTextField jTextField2 = null;**

**private JLabel jLabel3 = null;**

**private JTextField jTextField3 = null;**

**private JLabel jLabel4 = null;**

**private JComboBox jComboBox = null;**

**private JLabel jLabel5 = null;**

**private JTextField jTextField4 = null;**

**private JButton jButton = null;**

**private JPanel jPanel1 = null;**

**private JLabel jLabel6 = null;**

**private JTextField jTextField5 = null;**

**private JComboBox jComboBox1 = null;**

**private JButton jButton1 = null;**

**private JScrollPane jScrollPane = null;**

**private JTable jTable = null;**

**private JPopupMenu jPopupMenu = null;**

**private JMenuItem jMenuItem = null;**

**private String[] topics = { "by FirstName", "by Registration ID" };**

**private String[] accs = { "Current Account", "Savings Account", "Term Deposits" };**

**private DefaultTableModel list;**

**AccountHome home = null;**

**public static void main( String[] args ) {**

**new CustomerClient( "Aptech Bank - by BMP" );**

**}**

lookup từ xa EJB Account bằng JNDI

**public CustomerClient( String s ) {**

**super( s );**

**try {**

**Context ctx = new InitialContext();**

**Object ref = ctx.lookup( "AccountJNDI" );**

**home = ( AccountHome )PortableRemoteObject.narrow( ref, AccountHome.class );**

**} catch ( Exception e ) {**

**JOptionPane.showMessageDialog( jPanel, e.getMessage(),**

**"Error Message", JOptionPane.ERROR\_MESSAGE );**

**}**

**initialize();**

**setVisible( true );**

**}**

**private void initialize() {**

**this.setSize( 300, 256 );**

**this.setContentPane( getJContentPane() );**

**this.setResizable( false );**

**}**

**private JPanel getJContentPane() {**

**if ( jContentPane == null ) {**

**jContentPane = new JPanel();**

**jContentPane.setLayout( null );**

**jContentPane.add( getJTabbedPane(), null );**

**}**

**return jContentPane;**

**}**

**private JTabbedPane getJTabbedPane() {**

**if ( jTabbedPane == null ) {**

**jTabbedPane = new JTabbedPane();**

**jTabbedPane.addTab( "Registration",**

**new ImageIcon( getClass().getResource("/resource/images/search.gif") ),**

**getJPanel(), null );**

**jTabbedPane.addTab( "Search",**

**new ImageIcon( getClass().getResource("/resource/images/registry.gif") ),**

**getJPanel1(), null );**

**jTabbedPane.setBounds( 3, 0, 289, 221 );**

**jTabbedPane.addChangeListener( new javax.swing.event.ChangeListener() {**

**public void stateChanged( javax.swing.event.ChangeEvent e ) {**

**if ( jTabbedPane.getSelectedComponent() == jPanel )**

**initTabRegistration();**

**}**

**} );**

**}**

**return jTabbedPane;**

**}**

**private JPanel getJPanel() {**

**if ( jPanel == null ) {**

**jPanel = new JPanel();**

**jPanel.setLayout( null );**

**jPanel.add( getJLabel(), null );**

**jPanel.add( getJTextField(), null );**

**jPanel.add( getJLabel1(), null );**

**jPanel.add( getJTextField1(), null );**

**jPanel.add( getJLabel2(), null );**

**jPanel.add( getJTextField2(), null );**

**jPanel.add( getJLabel3(), null );**

**jPanel.add( getJTextField3(), null );**

**jPanel.add( getJLabel4(), null );**

**jPanel.add( getJComboBox(), null );**

**jPanel.add( getJLabel5(), null );**

**jPanel.add( getJTextField4(), null );**

**jPanel.add( getJButton(), null );**

**}**

**return jPanel;**

**}**

gọi phương thức Home qua home interface

gọi phương thức Home qua home interface

**private void initTabRegistration() {**

**try {**

**jTextField.setText( String.valueOf( home.getTopID() + 1 ) );**

**} catch ( Exception e ) { }**

**jTextField1.setText( "" );**

**jTextField2.setText( "" );**

**jTextField3.setText( "" );**

**jComboBox.setSelectedIndex( 0 );**

**jTextField4.setText( "" );**

**}**

**private JLabel getJLabel() {**

**if( jLabel == null ) {**

**jLabel = new JLabel();**

**jLabel.setSize( 106, 22 );**

**jLabel.setText( "Registration ID" );**

**jLabel.setLocation( 6, 5 );**

**}**

**return jLabel;**

**}**

**private JTextField getJTextField() { //** Registration ID, auto load

**if ( jTextField == null ) {**

**jTextField = new JTextField();**

**jTextField.setSize( 168, 22 );**

**jTextField.setLocation( 115, 5 );**

**try {**

**jTextField.setText( String.valueOf( home.getTopID() + 1 ) );**

**} catch ( Exception e ) { }**

**jTextField.setEnabled( false );**

**}**

**return jTextField;**

**}**

**private JLabel getJLabel1() {**

**if ( jLabel1 == null ) {**

**jLabel1 = new JLabel();**

**jLabel1.setSize( 106, 22 );**

**jLabel1.setText( "FirstName" );**

**jLabel1.setLocation( 6, 28 );**

**}**

**return jLabel1;**

**}**

**private JTextField getJTextField1() { //** First Name

**if ( jTextField1 == null ) {**

**jTextField1 = new JTextField();**

**jTextField1.setSize( 168, 22 );**

**jTextField1.setLocation( 115, 28 );**

**}**

**return jTextField1;**

**}**

**private JLabel getJLabel2() {**

**if ( jLabel2 == null ) {**

**jLabel2 = new JLabel();**

**jLabel2.setSize( 106, 22 );**

**jLabel2.setText( "LastName" );**

**jLabel2.setLocation( 6, 51 );**

**}**

**return jLabel2;**

**}**

**private JTextField getJTextField2() { //** Last Name

**if ( jTextField2 == null ) {**

**jTextField2 = new JTextField();**

**jTextField2.setSize( 168, 22 );**

**jTextField2.setLocation( 115, 51 );**

**}**

**return jTextField2;**

**}**

**private JLabel getJLabel3() {**

**if ( jLabel3 == null ) {**

**jLabel3 = new JLabel();**

**jLabel3.setSize( 106, 22 );**

**jLabel3.setText( "Address" );**

**jLabel3.setLocation( 6, 74 );**

**}**

**return jLabel3;**

**}**

**private JTextField getJTextField3() { //** Address

**if ( jTextField3 == null ) {**

**jTextField3 = new JTextField();**

**jTextField3.setSize( 168, 22 );**

**jTextField3.setLocation( 115, 74 );**

**}**

**return jTextField3;**

**}**

**private JLabel getJLabel4() {**

**if ( jLabel4 == null ) {**

**jLabel4 = new JLabel();**

**jLabel4.setSize( 106, 22 );**

**jLabel4.setText( "Account Type" );**

**jLabel4.setLocation( 6, 97 );**

**}**

**return jLabel4;**

**}**

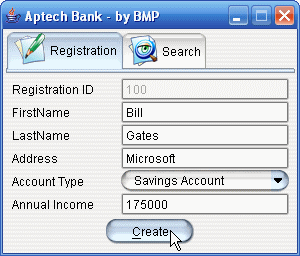
**private JComboBox getJComboBox() {**

**if ( jComboBox == null ) {**

**jComboBox = new JComboBox( accs );**

**jComboBox.setSize( 168, 22 );**

**jComboBox.setLocation( 115, 97 );**

 **}**

**return jComboBox;**

**}**

**private JLabel getJLabel5() {**

**if ( jLabel5 == null ) {**

**jLabel5 = new JLabel();**

**jLabel5.setSize( 106, 22 );**

**jLabel5.setText( "Annual Income" );**

**jLabel5.setLocation( 6, 120 );**

**}**

**return jLabel5;**

**}**

**private JTextField getJTextField4() {**

**if ( jTextField4 == null ) {**

**jTextField4 = new JTextField();**

**jTextField4.setSize( 168, 22 );**

**jTextField4.setLocation( 115, 120 );**

**}**

**return jTextField4;**

**}**

**private JButton getJButton() {**

**if ( jButton == null ) {**

**jButton = new JButton();**

**jButton.setBounds( 100, 146, 88, 25 );**

**jButton.setText( "Create" );**

**jButton.setMnemonic( KeyEvent.VK\_C );**

**jButton.addActionListener( new ActionListener() {**

**public void actionPerformed( ActionEvent e ) {**

**Validate.resetValidate();**

**if ( !Validate.isNotBlank( "FirstName", jTextField1.getText() )**

**|| !Validate.isNotBlank( "LastName", jTextField2.getText() )**

**|| !Validate.isNotBlank( "Address", jTextField3.getText() )**

**|| !Validate.isRealNum( "AnnualIncome", jTextField4.getText() ) )**

**{**

**JOptionPane.showMessageDialog( jPanel, Validate.errMessage,**

**"Error", JOptionPane.WARNING\_MESSAGE );**

**return;**

**}**

**int id = Integer.parseInt( jTextField.getText() );**

**String fName = jTextField1.getText().trim();**

gọi create() của home interface ⇒ INSERT một hàng dữ liệu mới vào bảng

**String lName = jTextField2.getText().trim();**

**String add = jTextField3.getText().trim();**

**String accType = ( String )jComboBox.getSelectedItem();**

**double anIncome = Double.parseDouble( jTextField4.getText() );**

**try {**

**Account acc = home.create( id, fName, lName, add, accType, anIncome );**

**JOptionPane.showMessageDialog( jPanel, "Account created!",**

**"Message", JOptionPane.INFORMATION\_MESSAGE );**

**initTabRegistration();**

**} catch ( Exception ex ) {**

**JOptionPane.showMessageDialog( jPanel, "The account exists!",**

**"Error Message", JOptionPane.ERROR\_MESSAGE );**

**}**

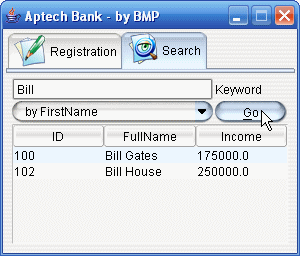
**}**

**} );**

**}**

**return jButton;**

**}**

 **private JPanel getJPanel1() {**

**if ( jPanel1 == null ) {**

**jPanel1 = new JPanel();**

**jPanel1.setLayout( null );**

**jPanel1.add( getJLabel6(), null );**

**jPanel1.add( getJTextField5(), null );**

**jPanel1.add( getJComboBox1(), null );**

**jPanel1.add( getJButton1(), null );**

**jPanel1.add( getJScrollPane(), null );**

**jPanel1.add( getJPopupMenu(), null );**

**}**

**return jPanel1;**

**}**

**private JLabel getJLabel6() {**

**if ( jLabel6 == null ) {**

**jLabel6 = new JLabel();**

**jLabel6.setSize( 72, 22 );**

**jLabel6.setText( "Keyword" );**

**jLabel6.setLocation( 209, 5 );**

**}**

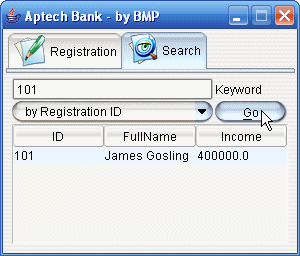
**return jLabel6;**

**}**

**private JTextField getJTextField5() { //** Keyword for search

**if ( jTextField5 == null ) {**

**jTextField5 = new JTextField();**

 **jTextField5.setBounds( 6, 5, 201, 22 );**

**}**

**return jTextField5;**

**}**

**private JComboBox getJComboBox1() {**

**if ( jComboBox1 == null ) {**

**jComboBox1 = new JComboBox( topics );**

**jComboBox1.setSize( 201, 22 );**

**jComboBox1.setLocation( 6, 28 );**

**}**

**return jComboBox1;**

**}**

**private JButton getJButton1() {**

**if ( jButton1 == null ) {**

**jButton1 = new JButton();**

**jButton1.setSize( 72, 22 );**

**jButton1.setLocation( 209, 28 );**

**jButton1.setText( "Go" );**

**jButton1.setMnemonic( KeyEvent.VK\_G );**

**jButton1.addActionListener( new ActionListener() {**

**public void actionPerformed( ActionEvent e ) {**

**Validate.resetValidate();**

**if ( !Validate.isNotBlank( "Keyword", jTextField5.getText() ) ) {**

**JOptionPane.showMessageDialog( jPanel1, Validate.errMessage,**

**"Error", JOptionPane.WARNING\_MESSAGE );**

**return;**

**}**

**String[] tmp = new String[3];**

**while ( list.getRowCount() > 0 ) list.removeRow( 0 );**

**try {**

**if ( ( ( String )jComboBox1.getSelectedItem() ).equals( topics[0] ) ) {**

**Collection c = null;**

**c = home.findByFirstName( jTextField5.getText().trim() );**

gọi **findByFirstName** của home interface ⇒ trả về Collection các Account

**if ( c != null ) {**

**Enumeration enumr = Collections.enumeration( c );**

**while ( enumr.hasMoreElements() ) {**

**Account acc = ( Account )enumr.nextElement();**

**AccountTO ato = acc.getAccountTO();**

**ato.setPrimaryKey( (AccountPK)acc.getPrimaryKey() );**

**tmp[0] = String.valueOf( ato.getId() );**

**tmp[1] = ato.getFirstName() + " " + ato.getLastName();**

**tmp[2] = String.valueOf( ato.getAnnualIncome() );**

**list.addRow( tmp );**

**}**

**}**

**}**

**if ( ( ( String )jComboBox1.getSelectedItem() ).equals( topics[1] ) ) {**

**AccountPK key = new AccountPK( Integer.parseInt( jTextField5.getText() ) );**

**Account acc = null;**

**acc = home.findByPrimaryKey( key );**

**if ( acc != null ) {**

**AccountTO ato = acc.getAccountTO();**

**tmp[0] = key.toString();**

gọi **findByPrimaryKey** của home interface ⇒ trả về một Account

**tmp[1] = ato.getFirstName() + " " + ato.getLastName();**

**tmp[2] = String.valueOf( ato.getAnnualIncome() );**

**list.addRow( tmp );**

**}**

**}**

**} catch ( Exception ex ) {**

**JOptionPane.showMessageDialog( jPanel1, "Account not found!", "Message",**

**JOptionPane.INFORMATION\_MESSAGE );**

**}**

**}**

**} );**

**}**

**return jButton1;**

**}**

**private JTable getJTable() {**

**if ( jTable == null ) {**

**list = new DefaultTableModel() {**

**public boolean isCellEditable( int rowIndex, int mColIndex ) {**

**return false;**

**}**

**};**

**list.addColumn( "ID" );**

**list.addColumn( "FullName" );**

**list.addColumn( "Income" );**

**jTable = new JTable( list );**

**jTable.setSelectionMode( ListSelectionModel.SINGLE\_SELECTION );**

**jTable.addMouseListener( new MouseAdapter() {**

**public void mouseReleased( MouseEvent e ) {**

**if ( e.isPopupTrigger() ) {**

**jTable.setRowSelectionInterval( jTable.rowAtPoint( e.getPoint() ),**

**jTable.rowAtPoint( e.getPoint() ) );**

**jPopupMenu.show( e.getComponent(), e.getX(), e.getY() );**

**}**

**}**

**public void mousePressed( MouseEvent e ) {**

**if ( e.isPopupTrigger() ) {**

**jTable.setRowSelectionInterval( jTable.rowAtPoint( e.getPoint() ),**

**jTable.rowAtPoint( e.getPoint() ) );**

**jPopupMenu.show( e.getComponent(), e.getX(), e.getY() );**

**}**

**}**

**});**

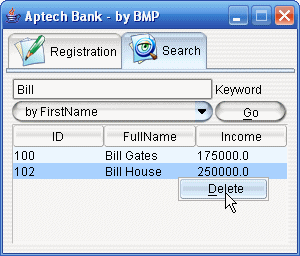
**}**

**return jTable;**

**}**

**private JScrollPane getJScrollPane() {**

**if ( jScrollPane == null ) {**

 **jScrollPane = new JScrollPane();**

**jScrollPane.setBounds( 6, 51, 276, 120 );**

**jScrollPane.setViewportView( getJTable() );**

**}**

**return jScrollPane;**

**}**

**private JPopupMenu getJPopupMenu() {**

**if ( jPopupMenu == null ) {**

**jPopupMenu = new JPopupMenu();**

**jPopupMenu.add( getJMenuItem() );**

**}**

**return jPopupMenu;**

**}**

**private JMenuItem getJMenuItem() {**

**if ( jMenuItem == null ) {**

**jMenuItem = new JMenuItem();**

**jMenuItem.setText( "Delete" );**

**jMenuItem.setMnemonic( KeyEvent.VK\_D );**

**jMenuItem.addActionListener( new ActionListener() {**

**public void actionPerformed( ActionEvent e ) {**

**String strId = jTable.getValueAt( jTable.getSelectedRow(), 0 ).toString();**

**if ( JOptionPane.showConfirmDialog( jPanel1,**

**"Are you want delete Account have AccountId is " + strId, "Delete",**

**JOptionPane.YES\_NO\_OPTION) == JOptionPane.NO\_OPTION )**

**return;**

**try {**

gọi **remove()** của home interface để xóa một hàng dữ liệu

**home.remove( new AccountPK( Integer.parseInt( strId ) ) );**

**} catch ( Exception ex ) { }**

**list.removeRow( jTable.getSelectedRow() );**

**}**

**});**

**}**

**return jMenuItem;**

**}**

**}**

- Lớp client trên cần có thêm lớp **Validate** để kiểm tra các ô nhập. Lớp **Validate** gồm các phương thức kiểm tra sử dụng regular expression[[3]](#footnote-4).

**package clients;**

**public class Validate {**

**public static String errMessage = "";**

**public static int errCount = 0;**

**public static void resetValidate() {**

**errMessage = "";**

**errCount = 0;**

**}**

**public static boolean isNotBlank( String objName, String sChar )**

**{**

**if ( sChar.trim().equals( "" ) ) {**

**errMessage += "\n" + objName + " must not blank";**

**errCount++;**

**return false;**

**}**

**return true;**

**}**

**public static boolean isRealNum( String objName, String sNum )**

**{**

**String RealPattern = "^\\d+(\\.\\d+)?$"; // Positive Real**

**if ( !sNum.matches( RealPattern ) ) {**

**errMessage += "\n" + objName + " is invalid Real number";**

**errCount++;**

**return false;**

**}**

**return true;**

**}**

**}**

**III. Triển khai trên JBoss 4.x**

1. Cấu hình cho một DataSource

- Trong ví dụ minh họa ta dùng hệ quản trị cơ sở dữ liệu là Microsoft SQL Server. Cần chuẩn bị **sqlserver-ds.xml**, (có tên bắt buộc là **\*-ds.xml**) dùng mô tả resource adapter đến MSSQL Server cho JBoss. Có nhiều driver dùng để kết nối với hệ quản trị cơ sở dữ liệu này:

+ Driver Type 1: cầu nối JDBC-ODBC, kết nối với nguồn dữ liệu đăng ký trong DSN (ODBC Data Source Name).

**<?xml version="1.0"?>**

**<datasources>**

**<local-tx-datasource>**

**<jndi-name>Bank</jndi-name>**

**<connection-url>jdbc:odbc:Bank</connection-url>**

**<driver-class>sun.jdbc.odbc.JdbcOdbcDriver</driver-class>**

**<user-name>sa</user-name>**

**<password />**

**</local-tx-datasource>**

**</datasources>**

+ Driver Type 4: driver Type 4 có được từ nhiều nhà cung cấp, sao chép vào thư mục **%JBOSS%\server\default\lib** các gói (.JAR) driver được chọn *trước* khi chạy server:

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

**<datasources>**

**<local-tx-datasource>**

**<jndi-name>Bank</jndi-name>**

**<connection-url>**

**jdbc:microsoft:sqlserver://venus:1433;databaseName=Bank**

**</connection-url>**

**<driver-class>com.microsoft.jdbc.sqlserver.SQLServerDriver</driver-class>**

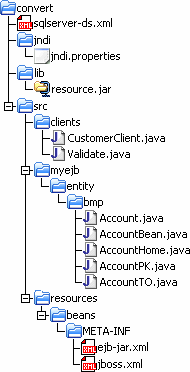
**<user-name>sa</user-name>**

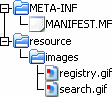
**<password></password>**

**</local-tx-datasource>**

**</datasources>**

2. Cấu trúc thư mục





**resource.jar**

3. Tạo và triển khai nhanh bằng Ant

- Chuẩn bị **build.xml**:

**<?xml version="1.0"?>**

**<project name="JBoss" default="ejbjar" basedir=".">**

**<property environment="env" />**

**<property name="src.dir" value="${basedir}/src" />**

**<property name="src.resources" value="${basedir}/src/resources" />**

**<property name="jboss.home" value="${env.JBOSS\_HOME}" />**

**<property name="build.dir" value="${basedir}/build" />**

**<property name="build.classes.dir" value="${build.dir}/classes" />**

**<property name="app.name" value="account" />**

**<!-- Build classpath -->**

**<path id="classpath">**

**<fileset dir="${jboss.home}/client">**

**<include name="\*\*/\*.jar" />**

**</fileset>**

**<pathelement location="${basedir}/lib/resource.jar" />**

**<pathelement location="${build.classes.dir}" />**

**<pathelement location="${basedir}/jndi" />**

**</path>**

**<!--**

0. Prepares the build directory

**============**

**-->**

**<target name="prepare">**

**<delete dir="${build.dir}" />**

**<mkdir dir="${build.dir}" />**

**<mkdir dir="${build.classes.dir}" />**

**</target>**

**<!--**

1. Compiles, bundles, deploys

**============**

**-->**

**<target name="compile" depends="prepare">**

**<javac srcdir="${src.dir}"**

**destdir="${build.classes.dir}"**

**debug="on"**

**deprecation="on"**

**optimize="off"**

**includes="\*\*">**

**<classpath refid="classpath" />**

**</javac>**

**</target>**

**<target name="ejbjar" depends="compile">**

**<jar jarfile="build/${app.name}.jar">**

**<fileset dir="${build.classes.dir}">**

**<include name="myejb/\*\*/\*.class" />**

**</fileset>**

**<fileset dir="${src.resources}/beans/">**

**<include name="\*\*/\*.xml" />**

**</fileset>**

**</jar>**

**<copy file="build/${app.name}.jar" todir="${jboss.home}/server/default/deploy" />**

**<copy file="sqlserver-ds.xml" todir="${jboss.home}/server/default/deploy" />**

**</target>**

**<!--**

2. Runs the tests

**============**

**-->**

**<target name="run.client">**

**<jar destfile="${build.dir}/client.jar">**

**<manifest>**

**<attribute name="Main-Class" value="CustomerClient" />**

**</manifest>**

**<fileset dir="${build.classes.dir}/clients">**

**<include name="\*.class" />**

**</fileset>**

**<zipfileset src="${basedir}/lib/resource.jar" />**

**</jar>**

**<java classname="clients.CustomerClient" fork="yes" dir=".">**

**<classpath refid="classpath" />**

**</java>**

**</target>**

**<!--**

3. Cleans up

**============**

**-->**

**<target name="clean">**

**<delete dir="${build.dir}" />**

**<delete file="${jboss.home}/server/default/deploy/${app.name}.jar" />**

**<delete file="${jboss.home}/server/default/deploy/sqlserver-ds.xml" />**

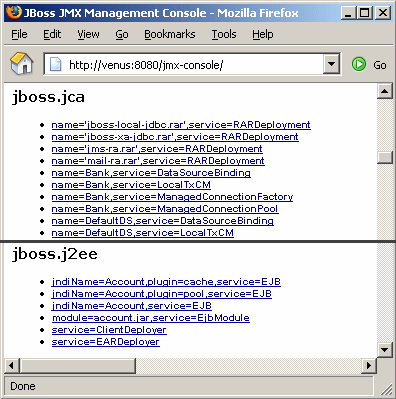
**</target>**

**</project>**

- Khởi động JBoss server: **%JBOSS\_HOME%\bin\run.bat**

- Biên dịch, đóng gói và triển khai EJB: **ant**

Kết quả triển khai trong jmx-console quản lý JBoss:



- Chạy client: **ant run.client**

**C. DAO (Data Access Object – đối tượng truy xuất dữ liệu)**

Client

Database2

**BMP**

Client

Tier

Bussiness

Tier

Integration

Tier

EIS

Tier

BMP Fields

*Network*

Tách Data Access Logic đưa vào Data Access Object (DAO)

Database1

**Refactoring 2**

Kết quả truy xuất từ DAO đóng gói trong Transfer Object

- Các thao tác truy xuất dữ liệu có thể được mã hóa bên trong các phương thức helper của BMP như đã mô tả. Để việc phát triển BMP chỉ tập trung vào business logic và không phụ thuộc vào cơ sở dữ liệu cần truy xuất, các thao tác truy xuất dữ liệu do các phương thức helper thực hiện được tách riêng ra và đóng gói vào trong *đối tượng truy xuất dữ liệu* DAO như một phần của bean. Điều này loại bỏ các thao tác truy xuất dữ liệu phức tạp khỏi bean. Dùng DAO cho phép nối kết đến nhiều mô hình dữ liệu khác nhau mà không cần thay đổi bean. Session bean và BMP đều có thể dùng DAO.

Dữ liệu lưu chuyển từ DAO đến bean có thể được đóng gói trong đối tượng Transfer Object.

- Khi một session bean hoặc một BMP truy xuất dữ liệu thông qua DAO, một phương thức tương ứng trong DAO sẽ cài đặt thực sự thao tác truy xuất (giống phương thức helper), bean chỉ cần gọi phương thức đó khi thực hiện các phương thức callback. Vì BMP thường dùng với những cơ sở dữ liệu có cách truy xuất phức tạp (có tập SQL query mở rộng, cách truy xuất đặc thù, … ), dùng DAO cho phép tách rời giữa business logic (trong bean) và thao tác truy xuất dữ liệu thực sự (trong DAO). Việc chọn DAO cho cơ sở dữ liệu tương ứng sẽ do người triển khai chọn lựa lúc triển khai, không cần thay đổi bean.

**BMP**

**DataAccessObject**

+ create:void

+ read:void

+ update:void

+ delete:void

<<TransferObject>>

**Data**

uses

creates

creates/uses

accesses

**DataSource**

**ResultSet**

creates

uses

1

1

1

\*

1. Giao diện DAO

- Bean sẽ gọi các phương thức truy xuất dữ liệu đóng gói trong DAO, được bộc lộ bởi giao diện DAO.

**package dao;**

**import myejb.entity.bmp.\* ;**

**import java.util.Collection ;**

**public interface AccountDAO {**

**public void create( int regId, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws AccountDAOException;**

**public void remove( AccountPK pk ) throws AccountDAOException;**

**public boolean findByPrimaryKey( AccountPK pk ) throws AccountDAOException;**

**public Collection findByFirstName( String firstName ) throws AccountDAOException;**

**public int selectTopID() throws AccountDAOException;**

**public AccountDetails load( AccountPK pk ) throws AccountDAOException;**

**public void store( int regId, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws AccountDAOException;**

**}**

2. Các lớp hỗ trợ DAO (tùy chọn)

- Các lớp này có thể là các lớp helper, Exception riêng của DAO.

**package dao;**

**public class AccountDAOException extends RuntimeException {**

**public AccountDAOException( String str ) {**

**super( str );**

**}**

**public AccountDAOException () {**

**super();**

**}**

**}**

3. Transfer Object cho DAO

- Mục đích dùng để chuyển tải dữ liệu của một hàng, đóng gói trong đối tượng Transfer Object.

**package dao;**

**public class AccountDetails {**

**private int id;**

**private String firstName;**

**private String lastName;**

**private String address;**

**private String accountType;**

**private double annualIncome;**

**AccountDetails( int id, String firstName, String lastName, String address,**

**String account, double annualIncome ) {**

**this.id = id;**

**this.firstName = firstName;**

**this.lastName = lastName;**

**this.address = address;**

**this.accountType = accountType;**

**this.annualIncome = annualIncome;**

**}**

**public String getFirstName() { return firstName; }**

**public String getLastName() { return lastName; }**

**public String getAddress() { return address; }**

**public String getAccountType() { return accountType; }**

**public double getAnnualIncome() { return annualIncome; }**

**}**

4. Lớp thực thi DAO

- Các phương thức truy xuất dữ liệu phức tạp (helper của bean) được tách khỏi bean và đóng gói trong DAO. So sánh với các phương thức helper tương ứng của bean để nhận thấy điều này.

**package dao;**

**import java.util.\*;**

**import myejb.entity.bmp.\*;**

**import java.sql.\*;**

**import javax.sql.\*;**

**import javax.naming.\*;**

**public class AccountDAOImpl implements AccountDAO {**

**private Connection con;**

**public void AccountDAOImpl() { }**

**public void create( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws AccountDAOException {**

**PreparedStatement pstmt = null;**

**try {**

**getConnection();**

**pstmt = con.prepareStatement( "INSERT INTO accounts VALUES ( ?, ?, ?, ?, ?, ? )" );**

**pstmt.setInt( 1, id );**

**pstmt.setString( 2, firstName );**

**pstmt.setString( 3, lastName );**

**pstmt.setString( 4, address );**

**pstmt.setString( 5, accountType );**

**pstmt.setDouble( 6, annualIncome );**

**pstmt.executeUpdate();**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException:" + e.getMessage() );**

**} finally {**

**closeStatement( pstmt );**

**closeDBConnection();**

**}**

**}**

**public void remove( AccountPK pk ) throws AccountDAOException {**

**int id = pk.getId();**

**PreparedStatement pstmt = null;**

**try {**

**getConnection();**

**pstmt = con.prepareStatement( "DELETE FROM Accounts WHERE id = ?" );**

**pstmt.setInt( 1, id );**

**pstmt.executeUpdate();**

**closeStatement( pstmt );**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException:" + e.getMessage() );**

**} finally {**

**closeStatement( pstmt );**

**closeDBConnection();**

**}**

**}**

**public boolean findByPrimaryKey( AccountPK pk ) throws AccountDAOException {**

**boolean result = false;**

**PreparedStatement pstmt = null;**

**try {**

**getConnection();**

**pstmt = con.prepareStatement( "SELECT \* FROM accounts WHERE id = ?" );**

**pstmt.setInt( 1, pk. getId() );**

**ResultSet rs = pstmt.executeQuery();**

**result = rs.next();**

**rs.close();**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException: "+ e.getMessage() );**

**} finally {**

**closeStatement( pstmt );**

**closeDBConnection();**

**}**

**return result;**

**}**

**public Collection findByFirstName( String firstName ) throws AccountDAOException {**

**Vector accounts = new Vector();**

**PreparedStatement pstmt = null;**

**try {**

**getConnection();**

**pstmt = con.prepareStatement( "SELECT \* FROM accounts WHERE firstName = ?" );**

**pstmt.setString( 1, firstName );**

**ResultSet rs = pstmt.executeQuery();**

**while ( rs.next() ) {**

**int id = rs.getInt( 1 );**

**accounts.addElement( new AccountPK( id ) );**

**}**

**rs.close();**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException: "+ e.getMessage() );**

**} finally {**

**closeStatement( pstmt );**

**closeDBConnection();**

**}**

**return accounts;**

**}**

**public int selectTopID() throws AccountDAOException {**

**int n = 99;**

**Statement stmt = null;**

**try {**

**getConnection();**

**stmt = con.createStatement();**

**ResultSet rs = stmt.executeQuery(**

**"SELECT TOP 1 id FROM accounts ORDER BY id DESC" );**

**if ( rs.next() ) n = rs.getInt( 1 );**

**rs.close();**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException: "+ e.getMessage() );**

**} finally {**

**closeStatement( stmt );**

**closeDBConnection();**

**}**

**return n;**

**}**

**public AccountDetails load( AccountPK pk ) throws AccountDAOException {**

**AccountDetails account = null;**

**PreparedStatement pstmt = null;**

**try {**

**getConnection();**

**pstmt = con.prepareStatement( "SELECT \* FROM accounts WHERE id = ?" );**

**pstmt.setInt( 1, pk getId() );**

**ResultSet rs = pstmt.executeQuery();**

**rs.next();**

**account = new AccountDetails( pk getId(), rs.getString( 2 ), rs.getString( 3 ),**

**rs.getString( 4 ), rs.getString( 5 ), rs.getDouble( 6 ) );**

**rs.close();**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException: "+ e.getMessage() );**

**} finally {**

**closeStatement( pstmt );**

**closeDBConnection();**

**}**

**return account;**

**}**

**public void store( int id, String firstName, String lastName, String address,**

**String accountType, double annualIncome ) throws AccountDAOException {**

**PreparedStatement pstmt = null;**

**try {**

**getConnection();**

**pstmt = con.prepareStatement**

**( "UPDATE accounts SET firstName = ?, lastName = ?, address = ?,"**

**+ " accountType = ?, annualIncome = ? WHERE id = ?" );**

**pstmt.setString( 1, firstName );**

**pstmt.setString( 2, lastName );**

**pstmt.setString( 3, address );**

**pstmt.setString( 4, accountType );**

**pstmt.setDouble( 5, annualIncome );**

**pstmt.setInt( 6, id );**

**pstmt.executeUpdate();**

**closeStatement( pstmt );**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException: "+ e.getMessage() );**

**} finally {**

**closeStatement( pstmt );**

**closeDBConnection();**

**}**

**}**

**private void getConnection() throws AccountDAOException {**

**try {**

**Context context = new InitialContext();**

**DataSource ds = ( DataSource ) context.lookup( "java:/Bank" );**

**con = ds.getConnection();**

**if ( con == null )**

**System.err.println( "Database Connection is null" );**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException:" + e.getMessage() );**

**} catch ( NamingException e ) {**

**throw new AccountDAOException("NamingException:" + e.getMessage() );**

**}**

**}**

**private void closeDBConnection() throws AccountDAOException {**

**try {**

**con.close();**

**} catch ( SQLException e ) {**

**throw new AccountDAOException( "SQLException:" + e.getMessage() );**

**}**

**}**

**// Statement** is superclass of **PreparedStatement**

**private void closeStatement( Statement stmt ) throws AccountDAOException {**

**try {**

**stmt.close();**

**} catch ( SQLException ex ) {**

**throw new AccountDAOException( "SQLException:" + ex.getMessage() );**

**}**

**}**

**}**

4. BMP dùng DAO

- BMP dùng DAO bao gồm: business logic, truy xuất cơ sở dữ liệu chỉ là các lời gọi đơn giản đến phương thức tương ứng trong giao diện DAO, sau đó truy xuất lại dữ liệu từ bean đơn giản **AccountDetails** nếu cần. So sánh với BMP không dùng DAO trên.

**package myejb.entity.bmp;**

**import dao.\*;**

**import javax.ejb.\*;**

**import java.sql.\*;**

**import javax.sql.\*;**

**import javax.naming.\*;**

**import java.util.\*;**

**public class AccountBean implements EntityBean {**

**protected EntityContext ctx;**

**//** Bean-managed state fields

**private int id;**

**private String firstName;**

**private String lastName;**

**private String address;**

**private String accountType;**

**private double annualIncome;**

**public AccountBean() {**

**System.out.println( "New BMP Object created by EJB Container" );**

**}**

**//** Business methods (Getter / Setter), Transfer Object

**public AccountTO getAccountTO() {**

**return new AccountTO( id, firstName, lastName, address, accountType, annualIncome );**

**}**

**public void setAccountTO( AccountTO updateObj ) {**

**this.id = updateObj.getId();**

**this.firstName = updateObj.getFirstName();**

**this.lastName = updateObj.getLastName();**

**this.address = updateObj.getAddress();**

**this.accountType = updateObj.getAccountType();**

**this.annualIncome = updateObj.getAnnualIncome();**

**}**

lookup đến DAO từ ENC thông qua JNDI

**//** Helper methods

**private AccountDAO getDAO() {**

**try {**

**AccountDAO dao = null;**

**Context context = new InitialContext();**

**dao = ( AccountDAO ) Class.forName**

**((String)context.lookup( "java:comp/env/param/AccountDAOClass" )).newInstance();**

**return dao;**

**} catch ( Exception e ) {**

**throw new EJBException( "getDAO:" + e.getMessage() );**

**}**

**}**

**//** Callback methods

**public AccountPK ejbCreate( int id, String firstName, String lastName,**

**String address, String accountType, double annualIncome ) throws CreateException {**

**System.out.println( "ejbCreate() called" );**

**try {**

**AccountDAO dao = getDAO();**

**dao.create( id, firstName, lastName, address, accountType, annualIncome );**

**this.id = id;**

**this.firstName = firstName;**

**this.lastName = lastName;**

**this.address = address;**

**this.accountType = accountType;**

**this.annualIncome = annualIncome;**

**return new AccountPK( id );**

**} catch ( Exception e ) {**

**throw new CreateException( "Failed to create Account " + e.getMessage() );**

**}**

**}**

**public void ejbPostCreate( int id, String firstName, String lastName,**

**String address, String accountType, double annualIncome ) { }**

**public void setEntityContext( EntityContext ctx ) { this.ctx = ctx; }**

**public void unsetEntityContext() { this.ctx = null; }**

**public void ejbRemove() throws RemoveException {**

**System.out.println( "ejbRemove() called" );**

**AccountPK pk = ( AccountPK )ctx.getPrimaryKey();**

**try {**

**AccountDAO dao = getDAO();**

**dao.remove( pk );**

**} catch ( Exception e ) {**

**throw new RemoveException( "Failed to remove Account " + e.getMessage() );**

**}**

**}**

**public void ejbActivate() {**

**System.out.println( "ejbActivate() called" );**

**}**

**public void ejbPassivate() {**

**System.out.println( "ejbPassivate() called" );**

**}**

**public void ejbLoad() {**

**System.out.println( "ejbLoad() called" );**

**AccountPK pk = (AccountPK)ctx.getPrimaryKey();**

**try {**

**AccountDAO dao = this.getDAO();**

truy xuất DAO điển hình:

- lookup thực thể DAO

- gọi phương thức DA logic tương ứng

- lấy dữ liệu trả về từ Transfer Object của DAO

**AccountDetails ato = dao.load( pk );**

**firstName = ato.getFirstName();**

**lastName = ato.getLastName();**

**address = ato.getAddress();**

**accountType = ato.getAccountType();**

**annualIncome = ato.getAnnualIncome();**

**} catch ( Exception e ) {**

**throw new EJBException( "ejbLoad: Unable to load from database", e );**

**}**

**}**

**public void ejbStore() {**

**System.out.println( "ejbStore() called" );**

**try {**

**AccountDAO dao = getDAO();**

**dao.store( id, firstName, lastName, address, accountType, annualIncome );**

**} catch ( Exception e ) {**

**throw new EJBException( "ejbStore: Unable to store into database", e );**

**}**

**}**

**//** Finder methods

**public AccountPK ejbFindByPrimaryKey( AccountPK key ) throws FinderException {**

**System.out.println( "ejbFindByPrimaryKey() called" );**

**try {**

**AccountDAO dao = getDAO();**

**boolean result = dao.findByPrimaryKey( key );**

**if ( !result )**

**throw new ObjectNotFoundException( "Account id " + key.getId() + " not found" );**

**return key;**

**} catch ( Exception e ) {**

**throw new FinderException( "FinderException: " + key. getId() + e.getMessage() );**

**}**

**}**

**public Collection ejbFindByFirstName( String firstName ) throws FinderException {**

**System.out.println( "ejbFindByFirstName() called" );**

**try {**

**AccountDAO dao = getDAO();**

**Collection c = dao.findByFirstName( firstName );**

**if ( c.isEmpty() ) throw new ObjectNotFoundException**

**( "Rows for firstname " + firstName + " not found" );**

**return c;**

**} catch ( Exception e ) {**

**throw new FinderException( "FinderException: " + firstName + e.getMessage() );**

**}**

**}**

**//** ejbHome methods

**public int ejbHomeGetTopID() throws FinderException {**

**System.out.println( "ejbHomeGetTopID() called" );**

**try {**

**AccountDAO dao = getDAO();**

**return dao.selectTopID();**

**} catch ( Exception e ) {**

**throw new FinderException( "FinderException: " + e.getMessage() );**

**}**

**}**

**}**

- Khi triển khai, deployment descriptor **ejb-jar.xml** cần có những thay đổi thích hợp để đăng ký lớp DAO vào ENC của ứng dụng trong cây JNDI.

**<ejb-jar>**

**<enterprise-beans>**

**<entity>**

**<ejb-name>Account</ejb-name>**

**<home>myejb.entity.bmp.AccountHome</home>**

**<remote>myejb.entity.bmp.Account</remote>**

**<ejb-class>myejb.entity.bmp.AccountBean</ejb-class>**

**<persistence-type>Bean</persistence-type>**

**<prim-key-class>myejb.entity.bmp.AccountPK</prim-key-class>**

**<reentrant>False</reentrant>**

**ENC**

**<env-entry>**

**<env-entry-name>param/AccountDAOClass</env-entry-name>**

**<env-entry-type>java.lang.String</env-entry-type>**

**<env-entry-value>dao.AccountDAOImpl</env-entry-value>**

**</env-entry>**

**<resource-ref>**

**<res-ref-name>Bank</res-ref-name>**

**<res-type>javax.sql.DataSource</res-type>**

**<res-auth>Container</res-auth>**

**</resource-ref>**

**</entity>**

**</enterprise-beans>**

1. POJO – Plain Old Java Object [↑](#footnote-ref-2)
2. Create, Read, Update, Delete [↑](#footnote-ref-3)
3. Xem về Regular Expression trong phần Tài liệu tham khảo. [↑](#footnote-ref-4)