Lab 09

Thực hành với DAO Pattern và JPA

Mục tiêu

- Thực hành với 1 framework cho phép truy xuất dữ liệu JPA - Java Persistence API
- Làm lại bài lab guide 8 với JPA
- Thực hành với Servlet Listener

Phần I Bài tập step by step

Bài 1.1

Mục tiêu:

- **Tạo các JPA POJO Object**
- **Sử dụng DAO Pattern**

Mở project eMarket. Trong lab 1, chúng ta sử dụng Entity Bean + Lookup JNDI để truy cập tới CSDL eMarket. Trong bài tập này chúng ta sẽ sử dụng JPA và hiển thị lại danh sách Product trên trang index.jsp

STEP 1: Xoá các class trong package bll và entity

Xoá package bll + ProductBLL

Xoá entity bean Product.

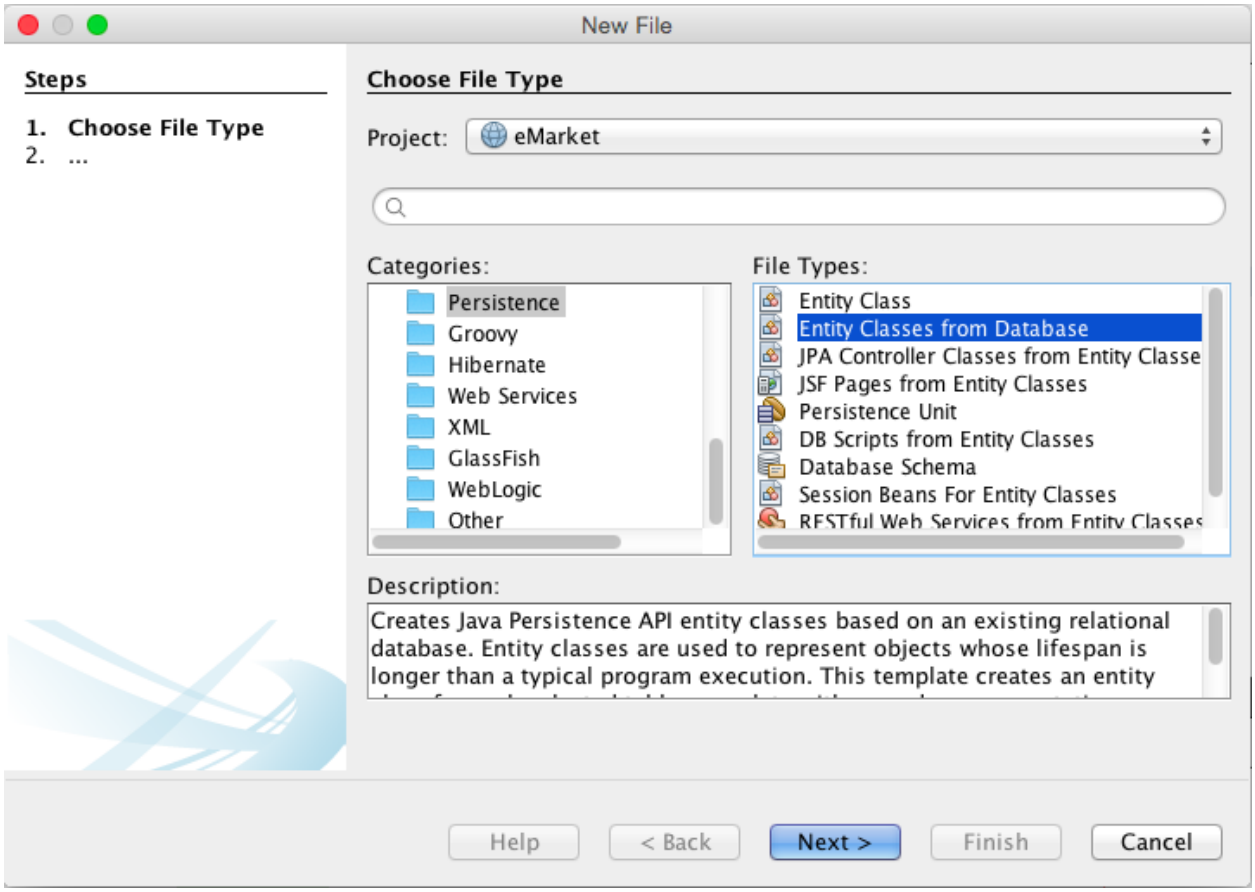
Xoá config liên quan đến resource trong file web.xml

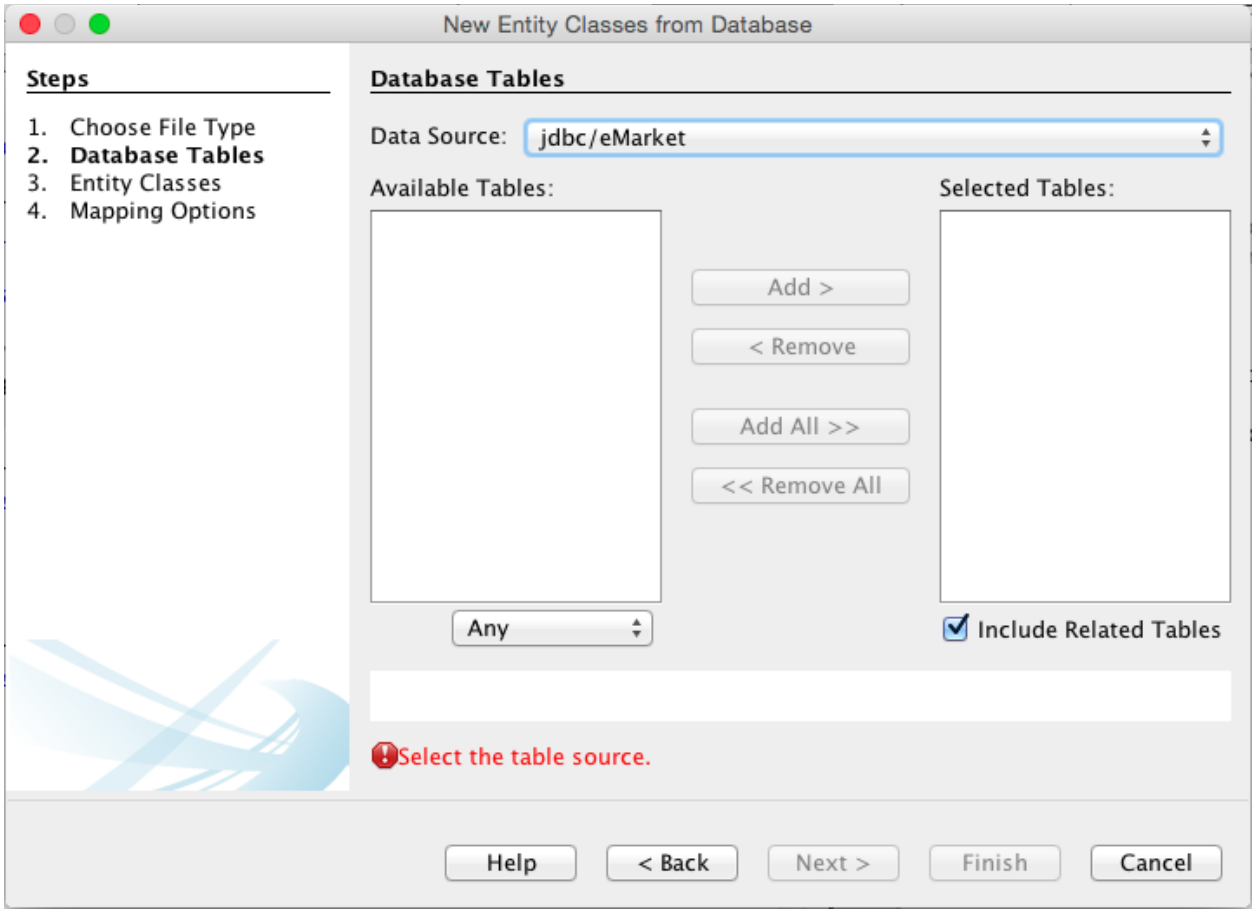
STEP 2: Tạo các class POJO cho các bảng của CSDL eMarket bằng Persistence.

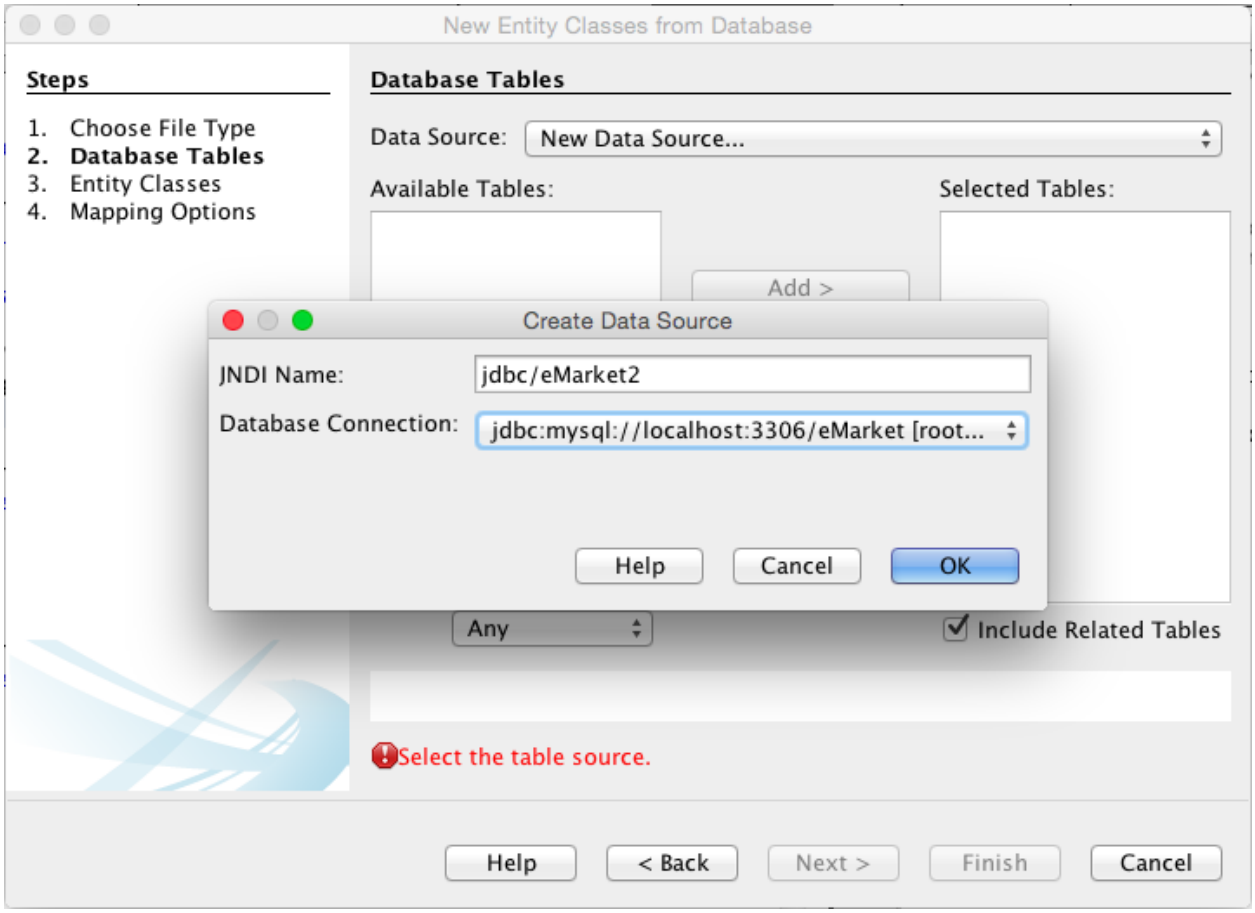
New -> Persistence -> Entity Classes from Database

Chọn Data Source: jdbc/eMarket từ Dropdown List.

Nếu các table không tự động hiện ra trong Available Tables thì chọn New Data Source.


New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project:
eMarket
Categories:
Persistence
Groovy
Hibernate
Web Services
XML
GlassFish
WebLogic
Other
File Types:
Entity Class
Entity Classes from Database
JPA Controller Classes from Entity Classe
JSF Pages from Entity Classes
Persistence Unit
DB Scripts from Entity Classes
Database Schema
Session Beans For Entity Classes
RESTful Web Services from Entity Classes
Description:
Creates Java Persistence API entity classes based on an existing relational database. Entity classes are used to represent objects whose lifespan is longer than a typical program execution. This template creates an entity
Help
< Back
Next >
Finish
Cancel


New Entity Classes from Database
Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. Mapping Options
Database Tables
Data Source:
jdbc/eMarket
Available Tables:
Selected Tables:
Add >
< Remove
Add All >>
<< Remove All
Any
Include Related Tables
Select the table source.
Help
< Back
Next >
Finish
Cancel



JNDI Name: jdbc/eMarket2

Database Connection: jdbc:mysql://localhost:3306/eMarket

Các bước để tạo Entity Class với JPA là:

- New Data Source
- Khai báo JNDI và connection
- Vào GlassFish, khai báo JNDI DataSource và ConnectionPool tương ứng.
- Sau đó các bảng của CSDL sẽ tự động bind vào để chúng ta generate các entity classes.

Trong trường hợp của chúng ta, jdbc/eMarket được list trong dropDown list vì nó có sẵn trên GlassFish (chúng ta đã tạo từ bài trước) nhưng không thể bind table được nên chúng ta sẽ tạo một JNDI khác.

Chọn tất cả các bảng để generate entity classes.

Mở file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="eMarketPU" transaction-type="JTA">
    <jta-data-source>jdbc/eMarket</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties/>
  </persistence-unit>
</persistence>
```

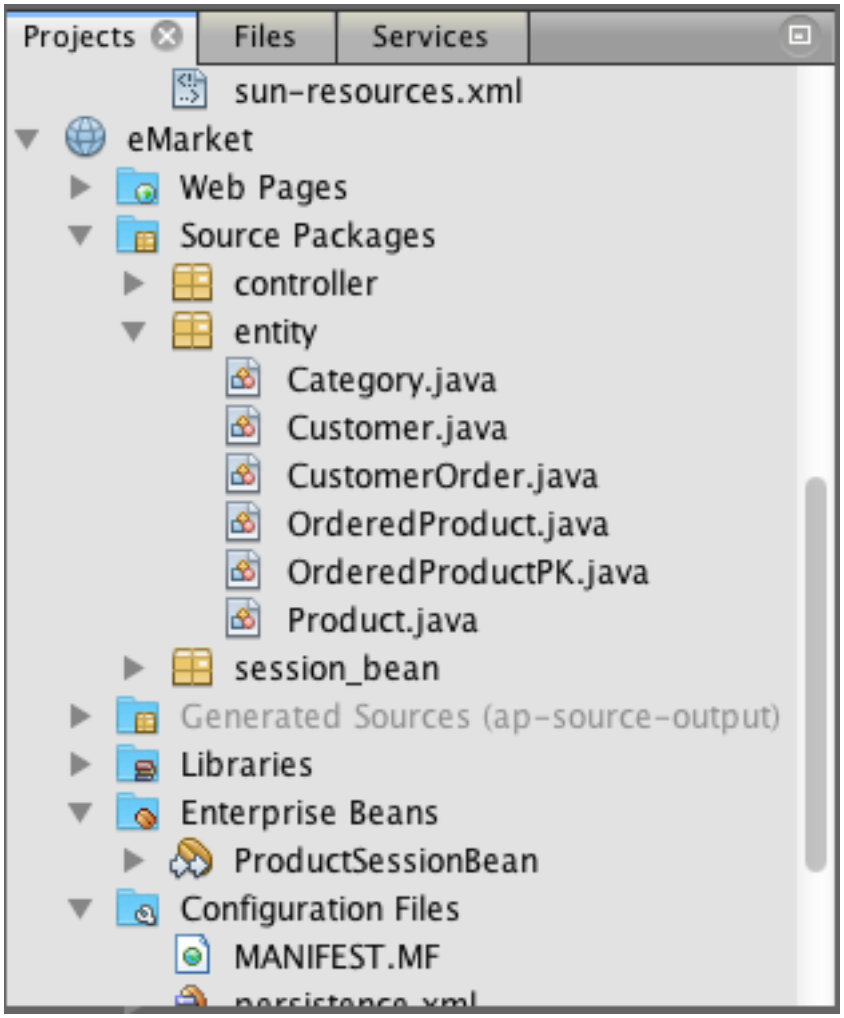
Sửa lại jta-data-source từ jdbc/eMarket2 thành jdbc/eMarket (để sử dụng tiếp JNDI jdbc/eMarket đã tạo trên GlassFish).

Mở file glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-
thread="false" connection-creation-retry-attempts="0" connection-creation-retry-
interval-in-seconds="10" connection-leak-reclaim="false" connection-leak-timeout-in-
seconds="0" connection-validation-method="auto-commit" datasource-
classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" fail-all-
connections="false" idle-timeout-in-seconds="300" is-connection-validation-
required="false" is-isolation-level-guaranteed="true" lazy-connection-
association="false" lazy-connection-enlistment="false" match-connections="false" max-
connection-usage-count="0" max-pool-size="32" max-wait-time-in-millis="60000"
name="mysql_eMarket_rootPool" non-transactional-connections="false" pool-resize-
quantity="2" res-type="javax.sql.DataSource" statement-timeout-in-seconds="-1"
steady-pool-size="8" validate-atmost-once-period-in-seconds="0" wrap-jdbc-
objects="false">
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="3306"/>
        <property name="databaseName" value="eMarket"/>
        <property name="User" value="root"/>
        <property name="Password" value="123456"/>
        <property name="URL" value="jdbc:mysql://localhost:3306/eMarket"/>
        <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/eMarket" object-type="user" pool-
name="MySQL/eMarket"/>
</resources>
```

Sửa lại jndi-name và pool-name tương ứng với JNDI DataSource đã có trên GlassFish.

Mở package entity, verify các entity classes được tạo ra như sau:



STEP 3: Xây dựng các session bean để thực hiện các chức năng business.

Sử dụng DAO pattern, chúng ta cần một data-access-object cho từng class entity. Trong bài này chúng ta sẽ tạo DAO cho class Product.

DAO của chúng ta là các session bean. Lợi ích của việc dùng session bean là chúng ta có thể chia sẻ các business-functionalities giữa các ứng dụng (bằng cách lookup EJB hoặc bằng Injection Dependency).

Tạo package session-bean -> Tạo abstract class AbstractSessionBean

Mỗi DAO của chúng ta sẽ extends abstract class này để tái sử dụng các business-logic chung (ví dụ như: Thêm, Sửa, Xoá, Tìm kiếm).

Class AbstractSessionBean.java:

```
package session_bean;
```

```
import java.util.List;
import javax.persistence.EntityManager;

/**
 *
 * @author ThanDieu
 */
public abstract class AbstractSessionBean<T> {

    private Class<T> entityClass;

    public AbstractSessionBean(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0]);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);
```

```
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }

}
```

Trong class này chúng ta implement các phương thức sau:

- create: insert them object vào bảng, kiểu entityClass là <T>
- edit: update
- remove: delete
- find, findAll, findRange: select theo các options

Tạo ProductSessionBean là một session bean và extends class abstract trên.

Chúng ta tạo session bean: stateless, no-interface. Vì chúng ta sử dụng bean này trong ứng dụng eMarket thôi. Nếu muốn share giữa các ứng dụng, các bạn cần Interface (local hoặc remote).

ProductSessionBean.java

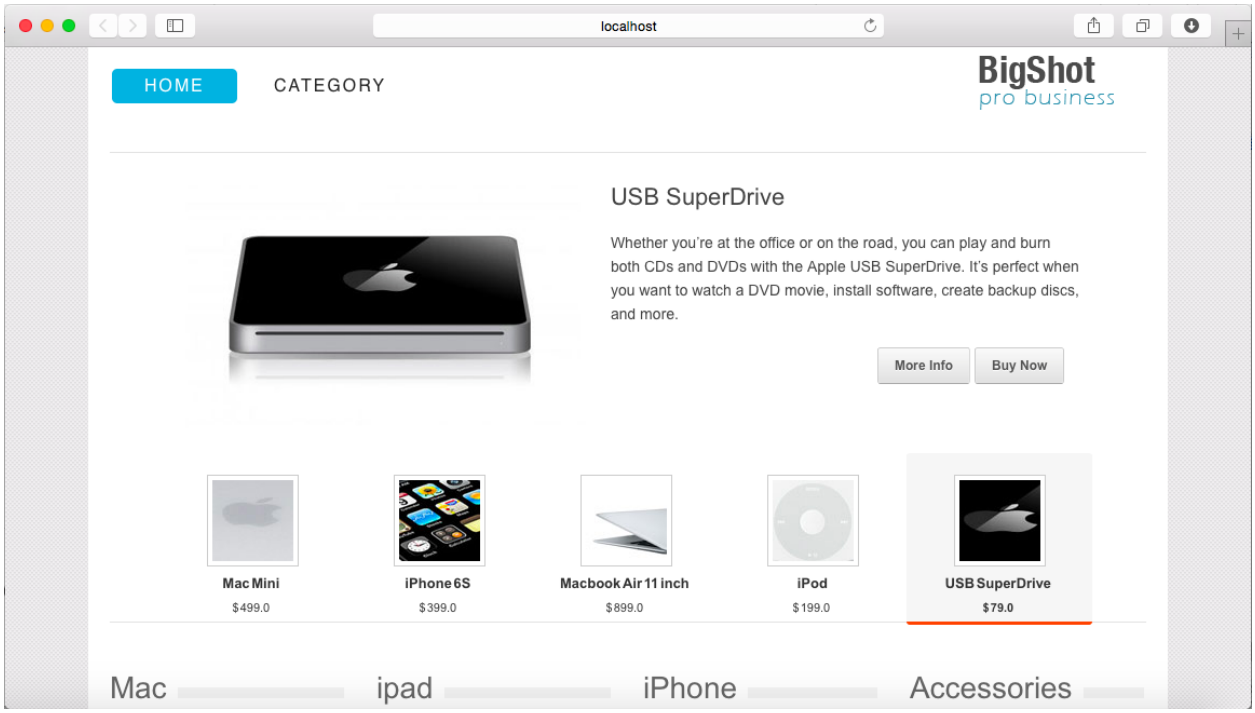
```
package session_bean;

import entity.Product;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author ThanDieu
 */
@Stateless
public class ProductSessionBean extends AbstractSessionBean<Product> {

    @PersistenceContext(unitName = "eMarketPU")
    private EntityManager em;

    protected EntityManager getEntityManager() {
        return em;
    }

    public ProductSessionBean(){
        super(Product.class);
    }
}
```

STEP 4: Trong ControllerServlet, sửa lại code của hàm init()

Inject dependency:

```
    @EJB
    private ProductSessionBean productSessionBean;

    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

        super.init(servletConfig);

        // store new product list in servlet context
        getServletContext().setAttribute("newProducts",
productSessionBean.findRange(new int[]{0,5}));
    }
```

Chạy chương trình:



Bài 1.2

Mục tiêu:

- Thực hành với Event Handling trong JSP và Servlet
- ServletContextListener

ServletContextListener được sử dụng khi chúng ta muốn một đoạn code được thực thi trước khi web application khởi động.

STEP 1: Thay thế đoạn code trong init() của ControllerServlet bằng cách thêm implement Servlet Context Listener.

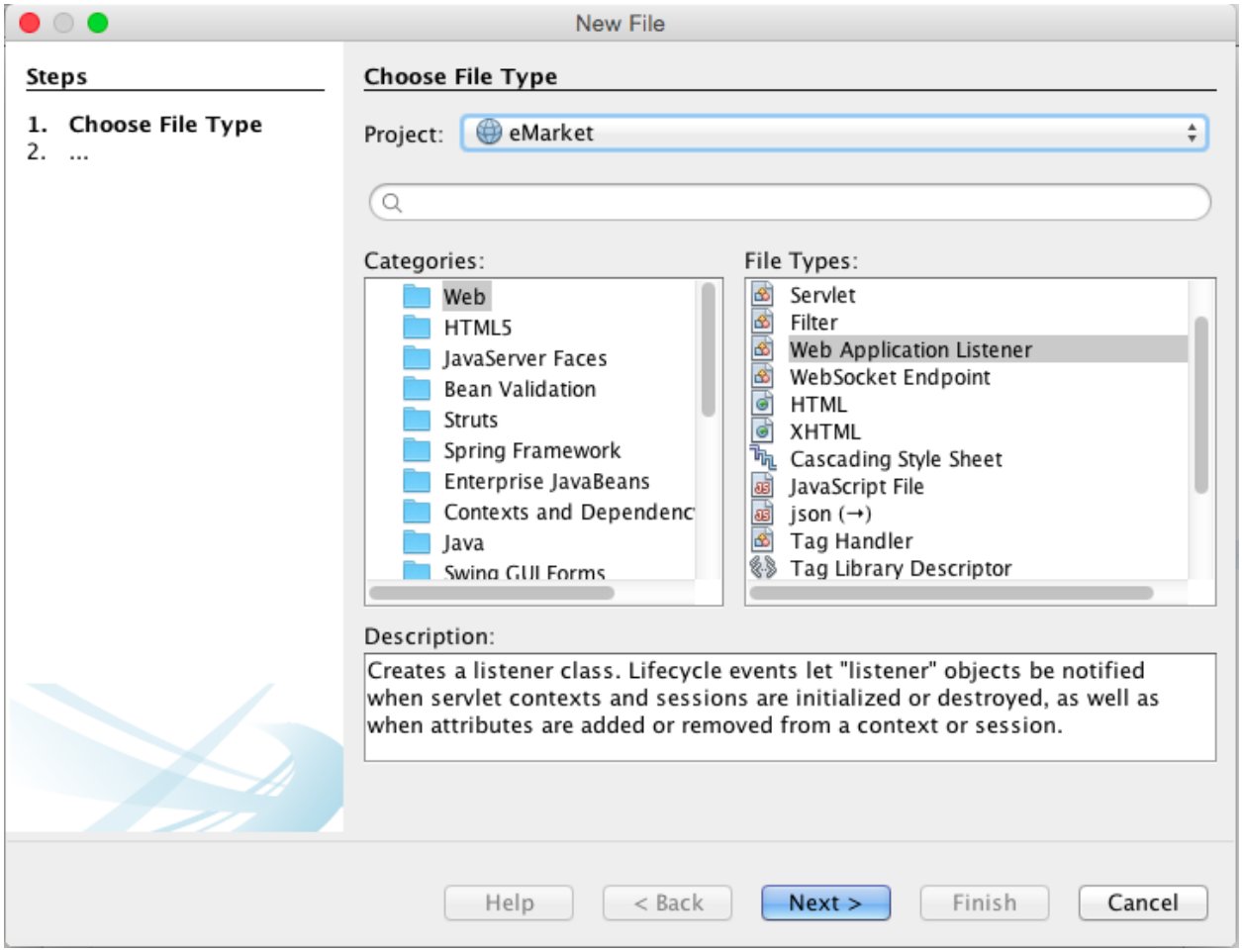
Mở project eMarket -> Mở Controller Servlet -> Bỏ tuỳ chọn loadOnStartup = 1 -> Bỏ hàm init().

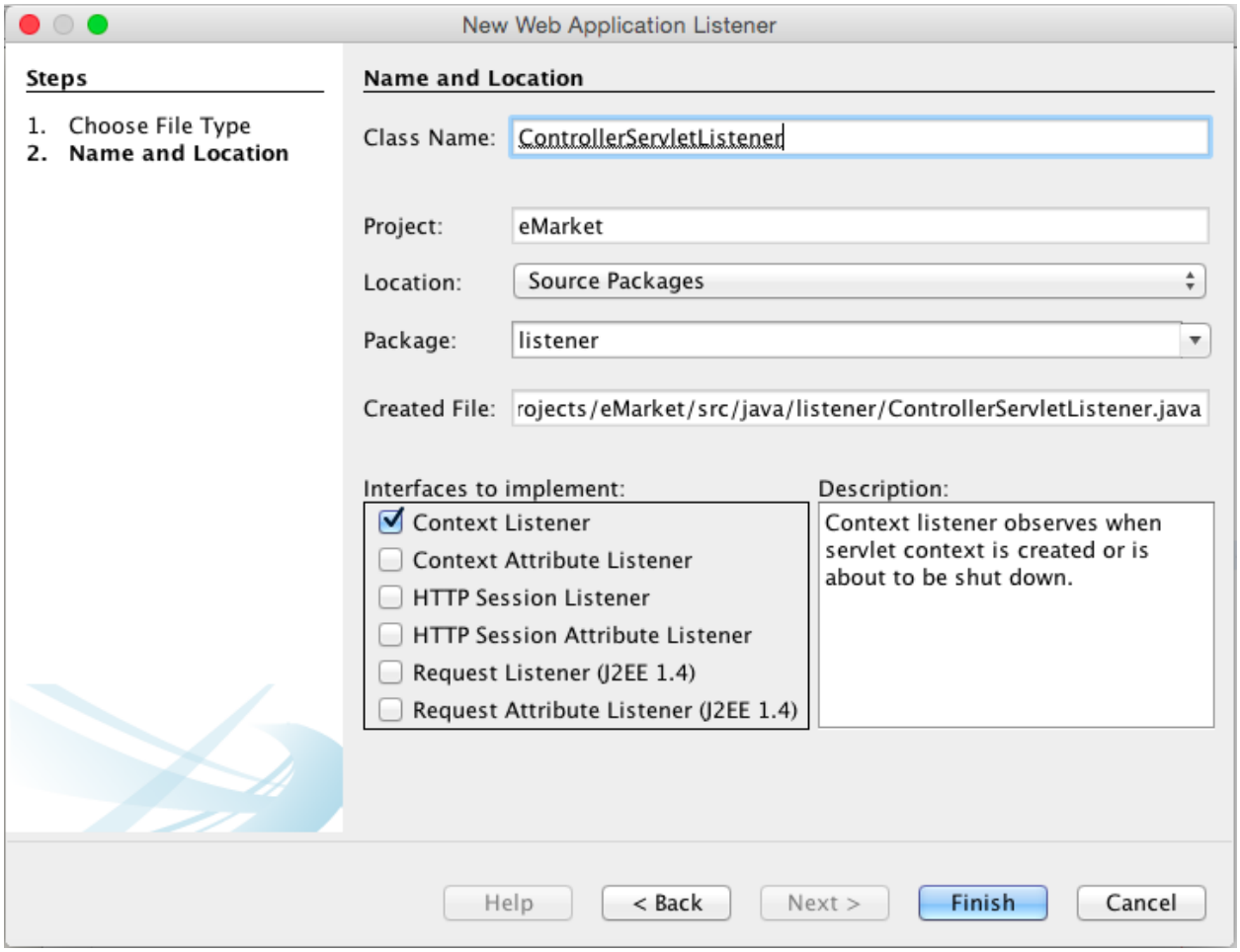
Class ControllerServlet như sau:

```
package controller;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author ThanDieu
 */
@WebServlet(name = "ControllerServlet",
            urlPatterns = {"/ControllerServlet"})
public class ControllerServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

    }
}
```

STEP 2: Tạo ControllerServletListener implements ServletContextListener

Tạo package listener -> Tạo ControllerListener implement ContextServletListener.


New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project:
eMarket
Categories:
Web
HTML5
JavaServer Faces
Bean Validation
Struts
Spring Framework
Enterprise JavaBeans
Contexts and Dependenc
Java
Swing GUI Forms
File Types:
Servlet
Filter
Web Application Listener
WebSocket Endpoint
HTML
XHTML
Cascading Style Sheet
JavaScript File
json (→)
Tag Handler
Tag Library Descriptor
Description:
Creates a listener class. Lifecycle events let "listener" objects be notified when servlet contexts and sessions are initialized or destroyed, as well as when attributes are added or removed from a context or session.
Help
< Back
Next >
Finish
Cancel



Mở Class ControllerServletListener, hai method cần phải được implement là contextInitialized và contextDestroyed.

Chúng ta sẽ lấy về một list các newProducts và gán attribute cho servlet context.

```
package listener;

import javax.ejb.EJB;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import session_bean.ProductSessionBean;

/**
 * Web application lifecycle listener.
 *
 * @author ThanDieu
 */
public class ControllerServletListener implements ServletContextListener {

    @EJB
    ProductSessionBean productSessionBean;
```

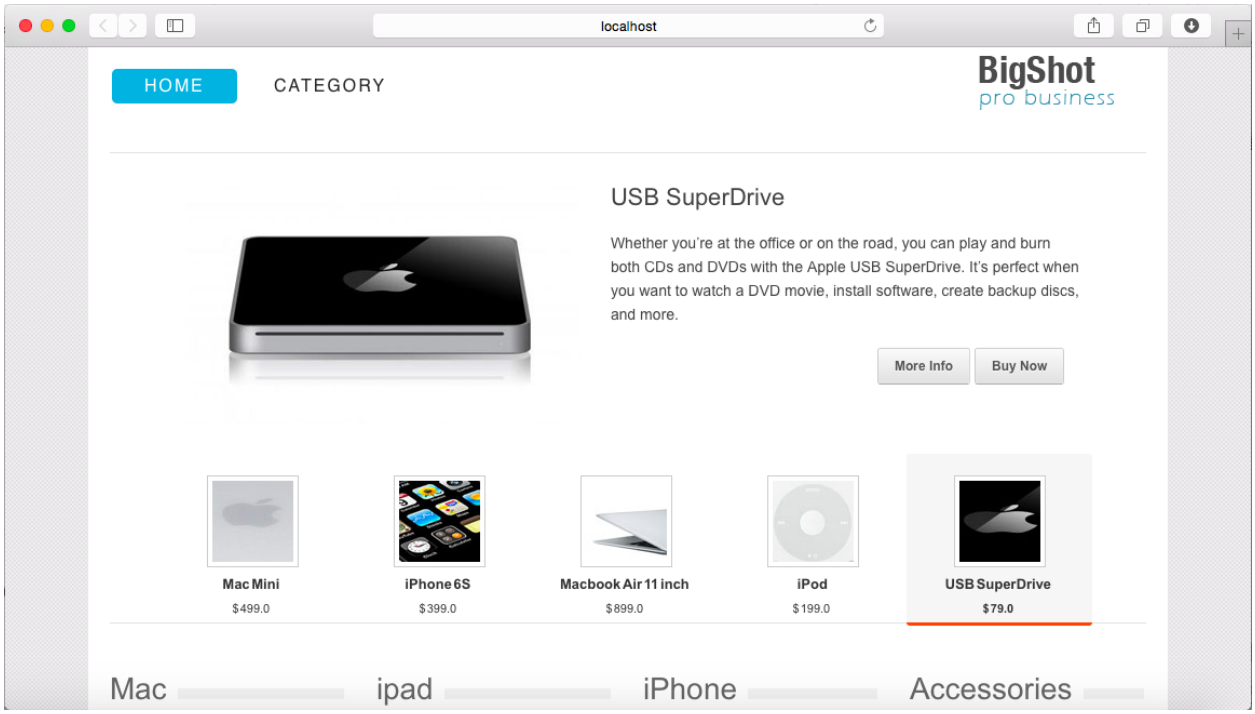
```
    ServletContext context;



    @Override
    public void contextInitialized(ServletContextEvent sce) {
        context = sce.getServletContext();
        context.setAttribute("newProducts", productSessionBean.findRange(new
int[] {0,5}));
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        this.context = null;
    }

}
```

Mở file web.xml để xem phần config lien quan đến listener vừa tạo:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <listener>
        <description>ServletContextListener</description>
        <listener-class>listener.ControllerServletListener</listener-class>
    </listener>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <jsp-config>
        <jsp-property-group>
            <description>JSP configuration for the store front</description>
            <url-pattern>/index.jsp</url-pattern>
            <url-pattern>/contact.jsp</url-pattern>
            <include-prelude>/WEB-INF/jspf/header.jspf</include-prelude>
            <include-coda>/WEB-INF/jspf/footer.jspf</include-coda>
        </jsp-property-group>
        <jsp-property-group>
            <description>JSP configuration for the admin
console</description>
            <url-pattern>/admin/*</url-pattern>
            <include-prelude>/WEB-INF/jspf/header.jspf</include-prelude>
            <include-coda>/WEB-INF/jspf/footer.jspf</include-coda>
        </jsp-property-group>
    </jsp-config>
    <context-param>
        <param-name>imgProductPath</param-name>
        <param-value>img/demo/</param-value>
    </context-param>
```

```
</web-app>
```

Chạy chương trình, kết quả phải giống bài 1.1



Phần 2 - Bài tập tự làm

Thêm danh sách các category vào project eMarket. Sử dụng JPA và ServletContextListener. (Làm lại phần bài tập tự làm trong Lab 08 sử dụng JPA và Servlet Context Listener).