Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина: Рис

**Отчёт**

К лабораторной работе №2

Выполнила: студента гр.814302

Мальцев Д.О.

Проверил: Тумаш Д.Е.

Минск 2020

**Цель:** Изучить принципы построения программы с помощью технологии Enterprise JavaBeans

**Теоретические сведения:**

**Enterprise JavaBeans** (также часто употребляется в виде аббревиатуры EJB) – спецификация технологии написания и поддержки [серверных](http://ru.wikipedia.org/wiki/Сервер_(программное_обеспечение)) компонентов, содержащих [бизнес-логику](http://ru.wikipedia.org/wiki/Бизнес-логика). Является частью [Java EE](http://ru.wikipedia.org/wiki/Java_EE).

Эта технология обычно применяется, когда бизнес-логика требует как минимум один из следующих сервисов, а часто все из них:

* поддержка сохранности данных (persistence); данные должны быть в сохранности даже после остановки программы, чаще всего достигается с помощью использования [базы данных](http://ru.wikipedia.org/wiki/База_данных)
* поддержка распределённых транзакций
* поддержка параллельного изменения данных и [многопоточность](http://ru.wikipedia.org/wiki/Многопоточность)
* поддержка событий
* поддержка именования и каталогов ([JNDI](http://ru.wikipedia.org/wiki/Java_Naming_and_Directory_Interface))
* безопасность и ограничение доступа к данным
* поддержка автоматизированной установки на [сервер приложений](http://ru.wikipedia.org/wiki/Сервер_приложений)
* удалённый доступ

Каждый EJB-компонент является набором [Java](http://ru.wikipedia.org/wiki/Java)-[классов](http://ru.wikipedia.org/wiki/Класс_(программирование)) со строго регламентированными правилами именования методов (верно для EJB 2.0, в EJB 3.0 за счет использования аннотаций выбор имён свободный). Бывают трёх основных типов:

* *объектные* (Entity Bean) – перенесены в спецификацию [Java Persistence API](http://ru.wikipedia.org/wiki/Java_Persistence_API)
* *сессионные* (Session Beans), которые бывают
  + stateless (без состояния)
  + stateful (с поддержкой текущего состояния сессии)
  + singleton (один объект на все приложение; начиная с версии 3.1)
* ***управляемые сообщениями* (Message Driven Beans) — их логика является реакцией на события в системе**

**Message-driven beans**

Так же как и session beans используются для**бизнесс-логики**. Отличие в том, что клиенты никогда не вызывают MDB напрямую. Обычно сервер использует MDB в **асинхронных** запросах.

**Entities**

**Entities** – это сущности каких то объектов и в EJB оно является хранилищем данных на период жизненного цикла Entity.

Entities является свое-родным отображением таблиц в БД.

Одним из главным достоинством EJB3 стал новый механизм работы с persistence, он дает возможность автоматически сохранять объекты в реляционной БД используя технологию ORM.

Для работы с entity был создан JPA (Java Persistence API).

**JPA определяет стандарт для**:

1) конфигурации маппинга сущностей приложения и их отображения в таблицах БД;

2) EntityManager API — позволяет выполнять CRUD (create, read, update, delete) операции над сущностями;

3) Java Persistence Query Language (JPQL) – для поиска и получения данных приложения;

**Реализация**  
Теперь рассмотрим реализацию этих сущностей.

В EJB3 мы используем **POJO**(Plain Old Java Objects), **POJI**(Plain Old Java Interfaces) и **аннотации**. Если с первыми двумя всё понятно, то про аннотации стоить поговорить отдельно. Аннотация записывается так:

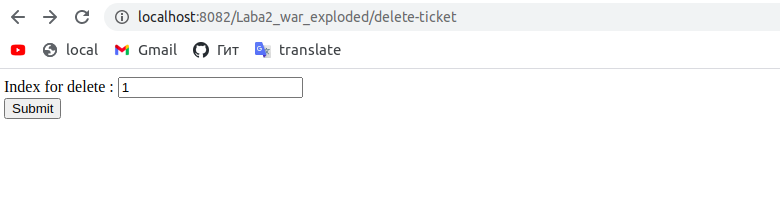
@<имя аннотации>(<список парамет-значение>)

**Задание**:13 вариант

1. Билеты (город отправления, прибытия, цена, класс)
   1. Вывести 2 билета с наименьшей ценой для выбранного класса

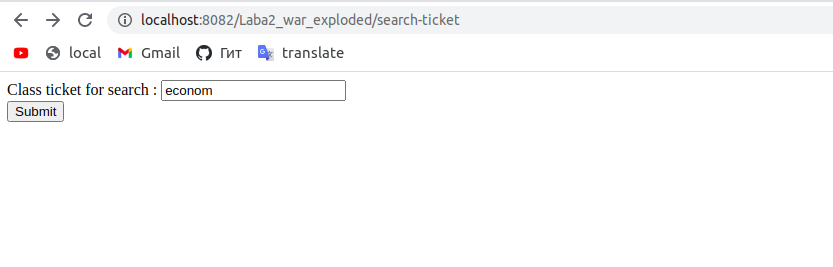
б. Удаление

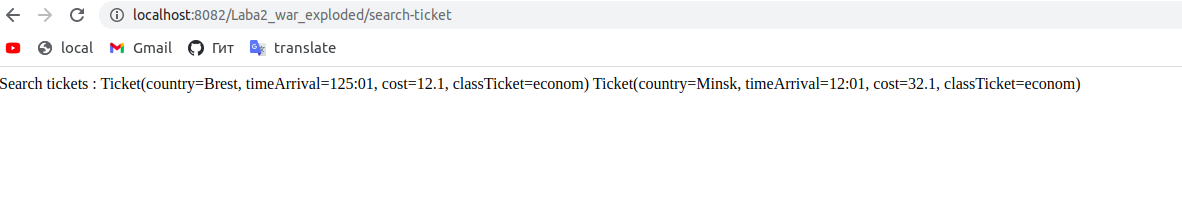
Работа программы : удаление





Реализация поиск :





Листинг кода :

package service;   
   
import model.Ticket;   
   
import javax.annotation.PostConstruct;   
import javax.ejb.Stateless;   
import java.util.Comparator;   
import java.util.LinkedList;   
import java.util.List;   
import java.util.stream.Collectors;   
   
   
@Stateless   
public class ActionWithTicketImpl implements ActionWithTicket {   
   
private List<Ticket> listTicket;   
   
@PostConstruct   
public void initListTicket() {   
listTicket = new LinkedList<>();   
listTicket.add(new Ticket("Minsk", "12:01", 32.1, "econom"));   
listTicket.add(new Ticket("Brest", "125:01", 12.1, "econom"));   
}   
   
@Override   
public String deleteTicket(int count) {   
return listTicket.remove(count).toString();   
}   
   
@Override   
public List<Ticket> searchTicket(String classTicket) {   
return listTicket.stream().   
filter(ticket -> ticket.getClassTicket().equals(classTicket))   
.sorted(new TicketComparator())   
.limit(2)   
.collect(Collectors.toList());   
   
}   
   
}   
   
class TicketComparator implements Comparator<Ticket> {   
   
public int compare(Ticket a, Ticket b) {   
return a.getCost().compareTo(b.getCost());   
}   
   
}

import service.ActionWithTicketImpl;   
   
import javax.ejb.EJB;   
import javax.servlet.ServletException;   
import javax.servlet.annotation.WebServlet;   
import javax.servlet.http.HttpServlet;   
import javax.servlet.http.HttpServletRequest;   
import javax.servlet.http.HttpServletResponse;   
import java.io.IOException;   
   
   
@WebServlet("/delete-ticket")   
public class DeleteServlet extends HttpServlet {   
   
@EJB   
ActionWithTicketImpl actionWithTicket;   
   
@Override   
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {   
request.getRequestDispatcher("delete-ticket.jsp").forward(request, response);   
}   
   
@Override   
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {   
   
request.setAttribute("ticket",actionWithTicket.deleteTicket(Integer.parseInt(request.getParameter("id"))));   
request.getRequestDispatcher("result-delete-ticket.jsp").forward(request, response);   
   
}   
} 

import model.Ticket;   
import service.ActionWithTicketImpl;   
   
import javax.ejb.EJB;   
import javax.servlet.ServletException;   
import javax.servlet.annotation.WebServlet;   
import javax.servlet.http.HttpServlet;   
import javax.servlet.http.HttpServletRequest;   
import javax.servlet.http.HttpServletResponse;   
import java.io.IOException;   
import java.util.List;   
   
@WebServlet("/search-ticket")   
public class SearchServlet extends HttpServlet {   
   
@EJB   
ActionWithTicketImpl actionWithTicket;   
   
@Override   
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {   
request.getRequestDispatcher("search-ticket.jsp").forward(request, response);   
}   
   
@Override   
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {   
String result="";   
List<Ticket> ticketList = actionWithTicket.searchTicket(request.getParameter("ticket"));   
   
for (Ticket ticket : ticketList) {   
result += ticket.toString() + "\n";   
}   
   
request.setAttribute("tickets",result);   
request.getRequestDispatcher("result-search-ticket.jsp").forward(request, response);   
}   
} 