МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра "ПОВТ и АС"

Java Data Base Connectivity и Java Persistence API

Методические указания к лабораторной работе
по дисциплинам "Объектно-ориентированное программирование"

Ростов-на-Дону 20 г.

Составитель: к.ф.-м.н., доц. Габрельян Б.В.
УДК 512.3

Java Data Base Connectivity и Java Persistence API: методические указания – Ростов н/Д: Издательский центр ДГТУ, 20 . – 8 с.

Рассмотрены технологии JDBC и JPA, схемы создания, и использования баз данных в Java-программах. Даны задания по выполнению лабораторной работы. Методические указания предназначены для студентов направлений 090304 "Программная инженерия", 020303 "Математическое обеспечение и администрирование информационных систем".

Ответственный редактор:

Издательский центр ДГТУ, 20

Данные постоянного хранения (persistence data) - это данные, хранящиеся на внешних устройствах. Данные могут храниться в двоичном или текстовом формате, могут содержать разметку, как, например, html-документы, могут иметь достаточно сложную логическую организацию, как, например, базы данных. Для работы с постоянно хранящимися данными разработаны разные Java-технологии. Далее рассматриваются два API, применяемые в основном для работы с базами данных: низкоуровневый, API подключения к базам данных (Java Database Connectivity - JDBC) и более высокоуровневый, поддерживающий объектно-реляционное отображение, Java Persistence API - JPA.

1. JDBC

Эта технология не является частью спецификации Java EE, но входит в Java SE. Дает возможность выполнять SQL-запросы к СУБД и получать от нее результаты обработки запросов. SQL-запрос в виде обычной строки передается методу стандартного JDBC-класса (Statement) и, если в результате запроса должны быть получены некоторые данные, метод возвращает их через объект другого стандартного JDBC-класса (ResultSet). Основные абстракции: драйвер, соединение (Connection), оператор (Statement), множество с результатами запроса (ResultSet).

1. Драйвер. JDBC-драйвер уникален для каждой конкретной СУБД и должен быть загружен с сайта разработчика этой СУБД (или с сайта разработчика драйвера для этой СУБД). Технически драйвер – это набор стандартных и собственных интерфейсов и классов, упакованный в Java- архив (jar-файл). По следующим ссылкам можно скачать JDBC- драйверы указанных СУБД:

Для PostgreSQL https://jdbc.postgresql.org/download.html

Для MySQL https://dev.mysql.com/downloads/connector/j/

Для Oracle https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html

Для MS SQL Server https://docs.microsoft.com/ru-ru/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server

О стороннем драйвере для SQL Lite можно прочитать здесь https://www.codejava.net/java-se/jdbc/connect-to-sqlite-via-jdbc

Для MS Access можно использовать сторонний драйвер http://ucanaccess.sourceforge.net/site.html

jar-файл драйвера должен быть доступен в нашем Java-приложении. Для этого его можно поместить либо в подкаталог lib нашего web-приложения, либо в подкаталог lib web-сервера. Например, если используется СУБД PostgreSQL и сервер Apache TomEE+ 8 (что предполагается в дальнейших примерах), скачан JDBC-драйвер **postgresql-42.3.1.jar** и TomEE развернут в папке c:\TomEE8, то нужно поместить файл postgresql-42.3.1.jar в каталог c:\TomEE\lib.

2. Соединение (интерфейс java.sql.Connection) связывает программу с конкретной базой данных. Чтобы установить соединение необходимо указать сервер базы данных, название базы данных, имя и пароль пользователя базы данных. Есть два основных способа получения соединения с базой данных: через объект класса java.sql.DriverManager (устаревший) или через интерфейс javax.sql.DataSource (рекомендуемый).

3. Оператор (интерфейс java.sql.Statement) – программный компонент, через который выполняются запросы к базе данных. Основные методы executeQuery() – запросы на получение информации (SQL-запрос SELECT), executeUpdate() – запросы на изменение базы данных (CREATE, INSERT, …), execute() – запросы как на получение информации, так и на изменение базы данных.

4. Результаты запроса (интерфейс java.sql.ResultSet), возвращается методом executeQuery(), интерфейса Statement. Основные методы boolean next(), Integer getInt(String attrName), String getString(String attrName), … ResultSet можно представлять себе как таблицу с данными в которой можно последовательно просматривать строки и для конкретной строки выбирать данные разных столбцов. Есть некий курсор, связанный с текущей, просматриваемой в данный момент строкой таблицы результатов. Изначально он связан с несуществующей строкой, строкой, расположенной перед первой строкой таблицы. Метод next() перемещает курсор на следующую строку и возвращает true, если это сделать удалось, то есть просмотрена еще не вся таблица, или false в противном случае. В текущей строке можно получить значение нужного столбца, указав имя этого столбца (или его номер, считая не с нуля, а с единицы) с помощью подходящего метода get (getInt, getString, … в зависимости от типа значений в столбце).

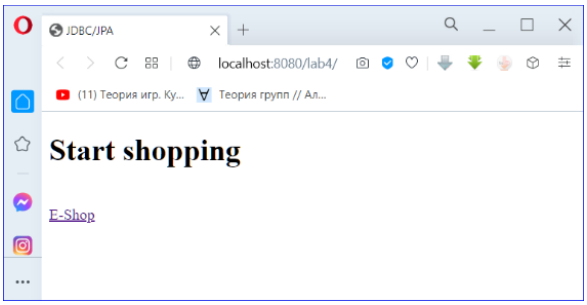
Для примера рассмотрим базу данных eshop, созданную в СУБД PostgreSQL для хранения информации, нужной электронному магазину по продаже автомобилей. База данных содержит несколько таблиц, но для примера нам достаточно знать, что в ней есть таблица CAR_TYPE, в которой хранится информация, связанная с типами автомобилей. В таблице есть три поля: 1) id – первичный ключ, целое число, назначаемое автоматически при создании новой записи (AUTO_INCTEMENT), 2) car_use – набор символов, задает тип – грузовой, легковой, …, 3) car_engine – набор символов, задает тип двигателя (использует бензин, газ, электричество, …). PostgreSQL и база eshop размещены на компьютере, используемом как сервер (например, на нашем персональном компьютере), на этом же компьютере размещен ApacheTomEE+ 8. Клиентские компьютеры (например, наш же персональный компьютер) через браузер (или клиентское приложение) посылают запросы, требующие

выборки из базы данных eshop и отображения результатов в браузере (или клиентском приложении) пользователя.

Проект будет состоять из стартовой jsp-страницы – index.jsp (View в архитектуре MVC) и сервлета, получающего запрос пользователя, обращающегося к базе данных eshop и генерирующего ответ пользователю (и Controler и Model и View в архитектуре MVC – хорошо ли это?).

index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head><title>JDBC/JPA</title></head>
<body>
<h1><%= "Start shopping" %></h1><br/>
<a href="e-shop">E-Shop</a>
</body>
</html>
```



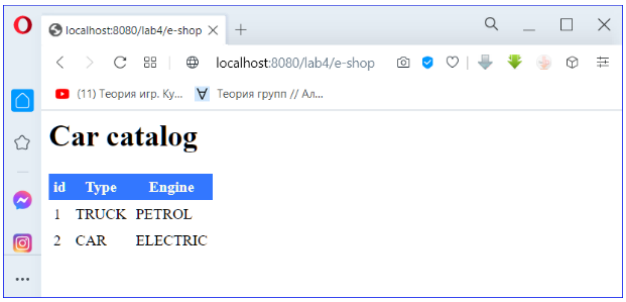
Рассмотрим последовательно оба варианта получения соединения: через DriverManager и через DataSource.

1.1. Соединение через DriverManager

```
import …;
import java.sql.*;

@WebServlet(name = "eshopServlet", value = "/e-shop")
public class Lab4 extends HttpServlet {
    private String url="jdbc:postgresql://localhost:5432/eshop";
    private String user="postgres";
    private String psw="xxx";
```

```
//
public void init() {
    try {
        Class.forName("org.postgresql.Driver");
    } catch(Exception e) { e.printStackTrace(); }
}
//
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
    response.setContentType("text/html");
    Connection con = null;
    Statement st = null;
    ResultSet res = null;
    //
    try {
        con = DriverManager.getConnection(url, user, psw);
        st = con.createStatement();
        res = st.executeQuery("SELECT id, car_use, car_engine FROM CAR_TYPE");
        //
        PrintWriter out = response.getWriter();
        out.println("<html><head>");
        out.println("<style type='text/css'>");
        out.println("TH { background: #3377ff; color: #fffff0; } ");
        out.println("TD, TH { border-right: 2px #000001; }");
        out.println("</style>");
        out.println("</head>");
```

```
            out.println("<body>");
            out.println("<h1>Car catalog</h1>");
            out.println("<table border='0' cellspacing='0' cellpadding='5'>");
            out.println("<tr><th>id</th><th>Type</th><th>Engine</th>");
            while (res.next()) {
                out.println("<tr>");
                out.println("<td>" + res.getInt("id") + "</td>");
                out.println("<td>" + res.getString("car_use") + "</td>");
                out.println("<td>" + res.getString("car_engine") + "</td>");
                out.println("</tr>");
            }
            out.println("</table>");
            out.println("</body></html>");
        } catch (SQLException sqle) {
            …
        } finally {
            try {
                if(res != null) res.close();
                if(st != null) st.close();
                if(con != null) con.close();
            } catch (SQLException se) {
                …
            }
        }
    }
}
```



Базу данных можно создать и из программы, например, следующий метод создает базу данных eshop, таблицу CAR_TYPE с полями id, car_use, car_engine в ней и добавляет в таблицу две записи:

```
public void createDB(Statement st) {
    try {
        st.executeUpdate("CREATE DATABASE ESHOP");
        st.executeUpdate(
            "CREATE TABLE CAR_TYPE " +
                "(id SERIAL PRIMARY KEY, " +
                "car_use VARCHAR(255), " +
                "car_engine VARCHAR(255))");
        st.executeUpdate("INSERT INTO CAR_TYPE VALUES (nextval('CAR_TYPE_id_seq'), 'TRUCK', 'PETROL')");
        st.executeUpdate("INSERT INTO CAR_TYPE VALUES (nextval('CAR_TYPE_id_seq'), 'CAR', 'ELECTRIC')");
    } catch(SQLException se) {
        …
    }
}
```

В данном примере вся учетная информация по соединению с базой данных размещается непосредственно в коде программы. Если поменяются какие-то параметры соединения, придется перекомпилировать программу. Очевидным решением является вынос всей учетной информации из программы во внешний файл. Стандартизованный подход для этого предлагает второй способ получения соединения – через DataSource.

1.2. Соединение через DataSource

Информация о соединении выносится в специальные файлы context.xml и web.xml.

META-INF/context.xml

```
<Context>
   <Resource name="jdbc/postgres" auth="Container"
         type="javax.sql.DataSource"
driverClassName="org.postgresql.Driver"
         url="jdbc:postgresql://127.0.0.1:5432/eshop"
         username="postgres" password="0" maxActive="20" maxIdle="10"
         maxWait="-1" />
</Context>
```

WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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_4_0.xsd"
       version="4.0">
   <resource-ref>
      <res-ref-name>jdbc/postgres</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
</web-app>
```

Учетная информация помещается в контекст приложения и оно может получить соединение с источником данных через класс, реализующий стандартный интерфейс javax.naming.Context. Собственно, источник данных (конкретная база данных) представлена в программе объектом класса, реализующего интерфейс javax.sql.DataSource. После получения соединения, как и в примере выше, работа с базой данных ведется через Statement.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

//
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
      response.setContentType("text/html");
      Connection con = null;
      Statement st = null;
      ResultSet res = null;
      //
      try {
            Context ic = new InitialContext();
            Context ctx = (Context)ic.lookup("java:comp/env");
            if(ic == null) {
               System.out.println("InitialContext error");
               return;
            }
            DataSource ds = (DataSource)ctx.lookup("jdbc/postgres");
```

```
            con = ds.getConnection();
            st = con.createStatement();
            res = st.executeQuery("SELECT id, car_use, car_engine FROM
CAR_TYPE");
            //
            PrintWriter out = response.getWriter();
            …
        } catch (Exception sqle) {
            …
        } finally {
            try {
                if(res != null) res.close();
                if(res != null) st.close();
                if(res != null) con.close();
            } catch (SQLException se) {
                …
            }
        }
    }
```

1.3. Внедрение зависимостей

Используем аннотации, позволяющие контейнеру создать нужный объект, представляющий, например, источник данных и связать его со ссылкой в программе (в классе). Тем самым зависимость программы от источника данных выносится из кода, а внешний агент (контейнер) внедряет в нее конкретный источник. Изменение характеристик источника (например, отдельные соединения или пул соединений) не приводит к каким-то

изменениям в самой программе. Для внедрения источника данных используется аннотация Resource, объявленная в пакете javax.annotation. Объявление этой аннотации в случае использования сервера Tomcat 9 содержится в файле annotations-api.jar. Он находится в каталоге lib. Однако такой файл отсутствует в каталоге lib TomEE 8. Этот файл нужно скопировать в каталог lib TomEE 8 (или в локальную папку lib проекта). В проекте нужно добавить ссылку на библиотеку annotations-api.jar. В IntelliJ Idea нужно выбрать File-Project Structure-Libraries, добавить библиотеку (+) annotations-api.jar.

В итоге объект DataSource уже не будет создаваться в программе.

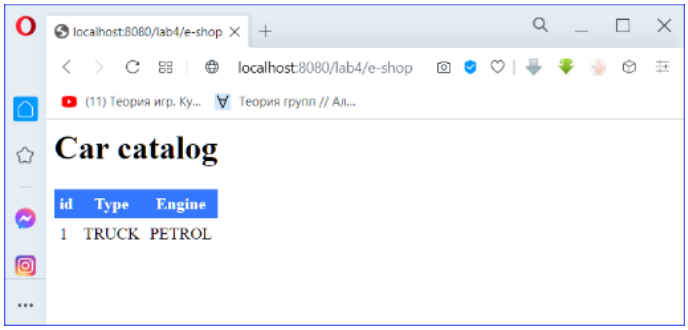
```
…
import javax.annotation.Resource;
…

@WebServlet(name = "eshopServlet", value = "/e-shop")
public class Lab4 extends HttpServlet {
        @Resource(name="jdbc/postgres")
        private DataSource ds;
//
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");
        Connection con = null;
        Statement st = null;
        ResultSet res = null;
        //
        try {
```

```
            con = ds.getConnection();
            st = con.createStatement();
            res = st.executeQuery("SELECT id, car_use, car_engine FROM
CAR_TYPE");
            //
            PrintWriter out = response.getWriter();
            …
        } catch (Exception sqle) {
            …
        } finally {
            try {
                if(res != null) res.close();
                if(res != null) st.close();
                if(res != null) con.close();
            } catch (SQLException se) {
                …
            }
        }
      …
}
```

1.3. PreperedStatement

Если некоторый запрос (или запросы) выполняется многократно, но с разными значениями параметров (в запросе заменяются значком '?'), нужно использовать PreparedStatement. Этот запрос компилируется и сохраняется в кэше, поэтому выполняется быстрее. Ссылка на PreparedStatement получается

также через соединение (Connection) при вызове метода prepareStatement. При этом должен быть задан сам запрос. Перед посылкой запроса должны быть заданы конкретные значения для параметров (если они есть). Для этого используются методы PreparedStatement: String getString(int paramNumber, String value), Integer getInt(int paramNumber, String value), …



paramNumber задает номер параметра (отсчитывается от 1).

Например,

…

Connection con = 0;

…

String sql = "SELECT id, car_use, car_engine FROM CAR_TYPE WHERE car_use like **?** and car_engine like **?**";

PreparedStatement pst = con.prepareStatement(sql);

pst.setString(1,"TRUCK");

pst.setString(2,"PETROL");

ResultSet rs = pst.**exequteQuery();**

…

2. JPA

В IntelliJ Idea Ultimate при создании нового проекта выбираем Persistence API. В папке META-INF будет создан файл persistence.xml. File-Project Structure-Libraries + annotation-api.jar. annotation-api.jar отсутствует в папке lib, но есть в папке lib Tomcat 9. Нужно скопировать в папку lib TomEE 8.

2.1. Сущности (Entities)

Класс Java (Plain Old Java Object - POJO), который:

1. Аннотирован @Entity или объявлен в дескрипторе как сущность;
2. Должен иметь конструктор по умолчанию (public или protected);
3. Должен быть внешним, не enum и не interface;
4. Не может быть final. Отображаемые поля и методы не могут быть final;
5. Serializable (для распределенных приложений);
6. Может быть abstract. Может расширять Entity и не Entity и наоборот;
7. Поля закрытые (доступны напрямую только из методов сущности);
8. Должен определять первичный ключ (аннотация @Id).

Сущность отображается на таблицу в базе данных. По умолчанию имя класса представляет имя таблицы, имена полей представляют имена столбцов таблицы (без учета регистра символов). Но можно задать другие имена используя аннотации.

Например,

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="car_type")
public class CarType {
    @Id
    @GeneratedValue
```

```
    private int id;

    @Column(name="car_use")
    private String use;

    @Column(name="car_engine")
    private String engine;

    public int getId() { return id; }
    public String getUse() { return use; }
    public void setUse(String t) { use = t; }
    public String getEngine() { return engine; }
    public void setEngine(String e) { engine = e; }
}
```

2.2. Менеджер сущностей (EntityManager)

Сущности отображаются на источник данных только тогда, когда они оказываются под управлением менеджера сущностей (EntityManager). Чтобы получить ссылку на объект EntityManager нужно определить "единицу сохраняемости" - persistence unit и обратиться к конкретному "провайдеру постоянства" (persistence provider). Для этого, во-первых, нужно прописать учетную информацию в файлах persistence.xml, context.xml (или resources.xml) и web.xml.

Файл **persistence.xml** (каталог resources/META-INF)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence 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_2.xsd"
        version="2.2">
      <persistence-unit name="eshopPU" transaction-
type="RESOURCE_LOCAL">
    <jta-data-source>jdbc/postgres</jta-data-source>
    <class>labs.lab4_2.CarType</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
            value="org.postgresql.Driver" />
      <property name="javax.persistence.jdbc.url"
            value="jdbc:postgresql://127.0.0.1:5432/eshop"/>
      <property name="javax.persistence.jdbc.user"
            value="postgres"/>
      <property name="javax.persistence.jdbc.password"
            value="xxx"/>
    </properties>
  </persistence-unit>
</persistence>
```

Кроме того, нужен файл **webapp/META-INF/context.xml**, такой же, как в параграфе 1.2. "Соединение через DataSource".

И, наконец, файл **WEB-INF/web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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_4_0.xsd" version="4.0">
   <resource-ref>
      <res-ref-name>eshopPU</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
</web-app>
```

Во-вторых, получить ссылку на EntityManager.

В java-приложении (сервлете)

```
import java.io.*;
import javax.persistence.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(name = "JPAServlet", value = "/eshop-servlet")
public class JPAEShopServlet extends HttpServlet {
        …
        private EntityManagerFactory emFactory;
        …
 @Override
 public void init() {
      emFactory = Persistence.createEntityManagerFactory("eshopPU");
 }
```

```
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");
        …
        EntityManager em = emFactory.createEntityManager();
        …
}
```

EntityManager позволяет осуществлять запросы к базе данных на поиск записей в указанной таблице (через представляющую ее сущность) с заданным значением первичного ключа. Например,

```
        …
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<table>");
        out.println("<tr><th>id</th><th>type</th><th>engine</th></tr>");
        out.println("<tr>");
        CarType ct = null;
        ct = em.find(CarType.class, i);
        if (ct != null) {
                out.println("<td>" + ct.getId() + "</td>");
                out.println("<td>" + ct.getUse() + "</td>");
                out.println("<td>" + ct.getEngie() + "</td>");
        }
        out.println("</tr>");
```

Более сложные запросы можно выполнять, используя язык запросов технологии Java Persistence (JPQL).

2.3. Язык запросов (Java Persistence Query Language - JPQL)

Похож на SQL, но вместо имен таблиц используются имена сущностей. Например, SQL-запроса

SELECT * FROM CAR_TYPE

на JPQL буте выглядеть так

SELECT c FROM CarType c

Запросы JPQL задаются в методах EntityManager:

Query createQuery(String jpqlRequest);

int executeUpdate(String jpqlRequest); // возвращает кол-во изменений

List getResultList();

Object getSingleResult();

В запросах можно задавать как конкретные значения, так и параметры, конкретные значения для которых нужно указывать после создания запроса, перед его выполнением. Тогда один и тот же запрос можно использовать многократно, задавая разные значения для параметра (параметров) запроса. Например,

…

Query q = em.createQuery(

"SELECT c FROM CarType c WHERE c.use LIKE **:val**"

);

Перед выполнением запроса параметру val должно быть назначено конкретное значение. Например,

q.setParameter("val", "TRUCK");

…

Запрос

List<CarType> ct = q.getResultList();

вернет список объектов CarType - результат запроса к базе данных, записей у которых значение в столбце car_use равно TRUCK.

ЗАДАНИЕ 1. Используйте какую-нибудь СУБД (например, PostgreSQL) для создания простейшей базы данных о некоторых товарах. Структура базы данных должна включать необходимые таблицы, например, "категория" (представляет категории товаров, например, комплектующие ПК, сетевое оборудование, компьютерная периферия, …), "производитель", "товар", "склад". Создайте web-приложение, позволяющее пользователю работать с информацией о товарах (или других объектах, в зависимости от того, какую информационную систему Вы создаете), хранящейся в базе данных (просматривать, добавлять, изменять, удалять), использующее для этого технологию JDBC.

ЗАДАНИЕ 2. Добавьте в приложение из задания 1 код, определяющий, создана ли нужная база данных. Если нет, ее структура должна создаваться самим приложением (а не средствами администрирования конкретной СУБД). Параметры соединения с базой данных должны передаваться приложению либо через дескриптор, либо через конфигурационный xml-файл.

ЗАДАНИЕ 3. Модифицируйте приложение, созданное Вами для реализации задания 1 лабораторной работы "Серверные страницы Java" так, чтобы данные хранились в базе данных. Вместо JDBC

используйте классы-сущности (Entity) и другие возможности технологии JPA.

ЗАДАНИЕ 4. Модифицируйте код из задания 3 так, чтобы получилось web-приложение, выполняющее аутентификацию пользователей и, в зависимости от роли пользователя, позволяющее либо задавать учетную информацию для пользователей, настройки соединения с базой данных (роль администратор), либо работать с товарами (или другими объектами, в зависимости от того, какую информационную систему Вы создаете), информация о которых хранится в базе данных. Учетная информация (логин и хеш пароля) должна храниться в базе данных или в xml-файле.

ЗАДАНИЕ 5. Если это не так, то организуйте программу для задания 4 так, чтобы она соответствовала архитектуре MVC.

Литература

1. JDBC Specification. Version 4.0. https://download.oracle.com/otndocs/jcp/jdbc-4.0-fr-eval-oth-JSpec/.
2. JDBC tutorial, https://docs.oracle.com/javase/tutorial/jdbc/basics/.
3. В.В.Смелов "Оновы web-программирования на Java" : учеб.-метод. пособие. – Минск: БГТУ. – 2009, 140 с.
4. Д.Хеффельфингер "Java EE 6 и сервер приложений GlassFish 3". – М.:ДМК-Пресс. – 2013, 416 с.
5. Jakarta EE. Jakarta Persistence. Version 3.0. Jakarta Persistence Team, https://jakarta.ee/specifications/persistence/. – 2020, 177 p.
6. Э.Гонсалвес "Изучаем Java EE 7". – СПб: Питер. – 2016, 640с.

7. К.Бауэр, Г.Кинг, Г.Грегори "Java Persistence API и Hibernate", 2-е изд. – М: ДМК Пресс. – 2019, 652с.

Редактор А.А. Литвинова

ЛР № 04779 от 18.05.01.	В набор	В печать
Объем 0,5 усл.п.л., уч.-изд.л.	Офсет.	Формат 60x84/16.
Бумага тип №3.	Заказ №	Тираж 120. Цена

Издательский центр ДГТУ

Адрес университета и полиграфического предприятия:

344010, г. Ростов-на-Дону, пл. Гагарина, 1.