Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра прикладного и системного программирования

С.А. Ермоченко

СРЕДСТВА И ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ JAVA

Методические рекомендации

Витебск
ВГУ имени П.М. Машерова
2023

УДК 004.432(075.8)
ББК 32.973.412я73+32.973.43я73
Е74

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 7 от 26.04.2023.

Автор: доцент кафедры прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **С.А. Ермоченко**

Р е ц е н з е н т ы :
заведующий кафедрой информационных систем и технологий
УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков;*
доцент кафедры информационных технологий и управления бизнесом
ВГУ имени П.М. Машерова,
кандидат педагогических наук, доцент *Н.Д. Адаменко*

Ермоченко, С.А.

Е74 Средства и технологии разработки web-приложений на платформе Java : методические рекомендации / С.А. Ермоченко. – Витебск : ВГУ имени П.М. Машерова, 2023. – 49 с.

В методических рекомендациях приведён подробный пошаговый пример разработки программного обеспечения простейшей информационной системы, имеющего формат web-приложения, запускаемого на популярной платформе Java. Пример сопровождается заданиями для выполнения лабораторных работ по закреплению пройденного материала.

Предназначаются для студентов первой ступени высшего образования специальностей: «Информационные системы и технологии» (дисциплины «Моделирование программного обеспечения», «Средства и технологии анализа и разработки информационных систем», «Современные технологии разработки web-приложений»); «Программная инженерия» (дисциплины «Объектно-ориентированные технологии программирования и стандарты проектирования», «Технологии взаимодействия с базами данных», «Web-технологии»); «Прикладная информатика» (дисциплина «Веб-программирование», «Программирование интернет-приложений»).

УДК 004.432(075.8)
ББК 32.973.412я73+32.973.43я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Информационные системы – это сложный комплекс обработки информации, состоящий (в узком смысле термина) из аппаратного обеспечения, программного обеспечения и обрабатываемых данных. В данном учебном издании уделяется внимание разработке программной части информационной системы.

При этом программное обеспечение может иметь различные формы (консольные, настольные, мобильные или web-приложения) и архитектурную организацию (одноуровневые, двухуровневые и многоуровневые). Наиболее популярными в настоящее время являются трёхуровневые информационные системы, состоящие из сервера баз данных, хранящих обрабатываемые данные; сервера приложений и клиентов. В качестве клиентов могут быть использованы толстый клиент (как правило, мобильное или настольное приложение, требующее отдельной установки на компьютере или устройстве конечного пользователя) или тонкий клиент (как правило, web-приложение, работающее в браузере и не требующее отдельной установки). Именно лёгкость запуска тонкого клиента, простота обновления программного обеспечения и высокая доступность сети Интернет сделали информационные системы на базе тонкого web-клиента так широко распространёнными.

В методических рекомендациях представлен пример разработки программного обеспечения информационной системы на базе платформы и технологий Java. Данная платформа и технология, а также язык программирования являются свободно распространяемыми, имеют длительную историю (с 1996-го года, что говорит о высокой надёжности и востребованности), отличаются высокой производительностью и кроссплатформенностью.

Помимо примера студентам предлагаются разноуровневые задания для лабораторных работ по закреплению полученных теоретических знаний на практике.

Материал соответствует темам учебных программ курсов: «Моделирование программного обеспечения», «Средства и технологии анализа и разработки информационных систем», «Современные технологии разработки web-приложений» (специальность «Информационные системы и технологии»); «Объектно-ориентированные технологии программирования и стандарты проектирования», «Технологии взаимодействия с базами данных», «Web-технологии» (специальность «Программная инженерия»); «Веб-программирование», «Программирование интернет-приложений» (специальность «Прикладная информатика»).

1. SERVLETS API И КОНТЕЙНЕР СЕРВЛЕТОВ TOMCAT

1.1. Постановка задачи

Разработаем простейшую информационную систему, позволяющую обрабатывать данные об одной сущности. Для того, чтобы не отвлекаться на особенности той или иной предметной области, будем работать со следующим entity-классом:

```
public class MyObject {
    private Integer id;
    private String fieldA;
    private Double fieldB;
    private Boolean fieldC;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getFieldA() {
        return fieldA;
    }

    public void setFieldA(String fieldA) {
        this.fieldA = fieldA;
    }

    public Double getFieldB() {
        return fieldB;
    }

    public void setFieldB(Double fieldB) {
        this.fieldB = fieldB;
    }

    public Boolean getFieldC() {
        return fieldC;
    }

    public void setFieldC(Boolean fieldC) {
        this.fieldC = fieldC;
    }
}
```

В нём описаны поля четырёх самых часто используемых типов: `Integer`, `Double`, `String` и `Boolean`.

В дальнейшем объекты данного класса будем сохранять в базу данных с таблицей аналогичной структуры.

1.2. Инструмент развёртывания приложений Maven

Разработка web-приложений требует решения множества сопутствующих технических проблем, специфичных именно для web-приложений. Для решения каждой их них программисты могут использовать некоторый набор готовых, хорошо зарекомендовавших себя сторонних библиотек. Все такие библиотеки, подключённые к проекту, и без которых уже невозможна работа приложения, принято называть внешними зависимостями. Компиляция и запуск приложений, которые могут иметь до нескольких сотен подобных зависимостей, представляет собой длительный рутинный процесс, предполагающий подключение библиотек или описание их местоположение в classpath для компилятора и виртуальной машины. Для автоматизации компиляции и запуска проекта, имеющего большое число зависимостей, существуют различные инструменты, одним из которых является Maven.

В IDE, поддерживающих разработку web-приложений на Java с использованием инструмента Maven, для этого, как правило, требуется создать проект специального типа, который так и называется Maven-проект. После создания такого проекта в корне проекта появляется файл настроек `pom.xml` (POM – Project Object Model, или объектная модель проекта). Для web-приложения после создания пустого проекта он может иметь следующий вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>by.vsu</groupId>
    <artifactId>web-application-example</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
       <maven.compiler.source>16</maven.compiler.source>
       <maven.compiler.target>16</maven.compiler.target>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <build>
       <plugins>
          <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-war-plugin</artifactId>
             <version>3.3.2</version>
          </plugin>
       </plugins>
    </build>
</project>
```

В данном файле пока нет никаких зависимостей. Их мы добавим позже. Помимо созданного в корне проекта файла `pom.xml`, Maven-проект имеет ещё несколько отличий от классических Java-проектов. Так, в Maven проекте используется немного другая структура каталогов. Исходный код храниться не просто в каталоге `src`, он разбивается на несколько каталогов:

`src\main\java` – каталог для хранения исходного кода основных классов приложения.

`src\main\resources` – каталог для хранения ресурсов, необходимых основным классам приложений. Как правило это некоторые properties-файлы или XML-файлы, используемые теми или иными библиотеками. Например, файл настроек инструмента журналирования log4j: `log4j2.xml`.

`src\main\webapp` – каталог с содержимым, которое должно размещаться в контейнере сервлетов. Об этом каталоге будет рассказано ниже.

`src\test\java` – каталог для хранения исходного кода классов, реализующих модульные тесты.

`src\test\resources` – каталог для хранения ресурсов, необходимых классам, реализующим модульные тесты. Например, файлы настроек инструментов, применяемых для изоляции модульных тестов, или инструментов для генерации объектов-заглушек.

Таким образом, после создания Maven-проекта, можно создать описанный выше класс `MyObject` в каталоге `src\main\java`.

1.3. Контейнер сервлетов

Для запуска web-приложений, разработанных с использованием Java-технологий, можно использовать готовый web-сервер, который умеет взаимодействовать со специальными Java-классами. Для такого взаимодействия разработан специальный набор интерфейсов, входящих в поставку Java Enterprise Edition. Основным из этих интерфейсов является интерфейс `javax.servlet.Servlet`. По имени этого интерфейса все классы, его реализующие, также принято называть сервлетами. А web-приложения, которые умеют запускать такие сервлеты и взаимодействовать с ними через указанный интерфейс, принято называть контейнерами сервлетов. Сам набор интерфейсов обычно называют Servlets API.

Для запуска нашего примера будем использовать контейнер сервлетов Apache Tomcat (`https://tomcat.apache.org/`). Данный web-сервер представляет собой обычное консольное приложение, написанное на Java.

Для запуска сервера необходимо установить переменную окружения `JAVA_HOME`, указывающую на папку, в которой установлена JDK. Пусть JDK установлена в папку `C:\Program Files\Java\jdk16`. Тогда можно

использовать следующую команду в консоли для установки значения этой переменной окружения:

```
set JAVA_HOME=C:\Program Files\Java\16
```

Также для запуска сервера необходимо после распаковки архива с сервером в некоторую папку установить переменную окружения CATALINA_HOME, указывающую на эту папку. Пусть сервер распакован в папку C:\tomcat. Тогда можно использовать следующую команду в консоли для установки значения этой переменной окружения:

```
set CATALINA_HOME=C:\tomcat
```

После чего для запуска сервера можно использовать следующую команду:

```
%CATALINA_HOME%\bin\startup.bat
```

Для остановки сервера можно использовать следующую команду:

```
%CATALINA_HOME%\bin\shutdown.bat
```

После запуска сервера в браузере будет доступна панель управления сервером по адресу http://localhost:8080/. Для доступа к панели управления нужно ввести имя и пароль пользователя, которые должны быть прописаны в файле %CATALINA_HOME%\conf\tomcat-users.xml. Предположим, что мы описали пользователя в этом файле следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
    <role rolename="manager"/>
    <user username="root" password="12345" roles="manager"/>
</tomcat-users>
```

Тогда при открытии панели управления (после нажатия на кнопку «Manager App») необходимо будет в открывшемся окне ввести имя пользователя root и пароль 12345.

Для развёртывания приложений на сервере Tomcat, можно добавить подкаталог в специальном каталоге сервера %CATALINA_HOME%\webapps. Но при этом возникает необходимость постоянного копирования необходимых файлов их каталога проекта, созданного в IDE, в каталог на сервере. При работе с удалённым сервером (работающим на другом компьютере), так и приходится поступать. Есть много различных путей автоматизации развёртывания проекта на удалённом сервере, но эти вопросы выходят за рамки данного методического пособия. Решением подобных вопросов занимаются DevOps-инженеры. Разработчик же, как правило, разворачивает проект у себя локально. В таком случае есть возможность настроить Tomcat таким образом, чтобы он запускал проект не из своего внутреннего каталога webapps, а из другого указанного места.

Предположим, что наш проект был создан в каталоге `D:\java-projects\web-application-example`. При запуске процедуры сборки проекта с помощью Maven (будем использовать команду Maven `package`), в каталоге проекта создаётся подкаталог `target`, внутри которого будет находится ещё один подкаталог `web-application-example-1.0-SNAPSHOT`. Полный путь к этому каталогу будет следующий:

`D:\java-projects\web-application-example\target\web-application-example-1.0-SNAPSHOT`

Содержимое этого каталога полностью соответствует требованиям контейнера сервлета по размещению приложений на сервере. Поэтому можно либо полностью копировать содержимое этого каталога в некоторый подкаталог каталога `%CATALINA_HOME%\webapps`. Либо настроить Tomcat на запуск приложения из этого каталога. Для этого в каталоге сервера `%CATALINA_HOME%\conf\Catalina\localhost` необходимо добавить файл с именем, соответствующем имени приложения `web-application-example.xml` со следующим содержимым:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/web-application-example"
         docBase="D:\java-projects\web-application-example\target\web-application-example-1.0-SNAPSHOT"/>
```

После чего необходимо перезапустить сервер.

1.4. Структура web-приложения на сервере

При развёртывании web-приложения на сервере в папке приложения должна быть создана строго определённая структура подкаталогов. Эта структура в папке проекта в IDE будет формироваться внутри каталога `src\main\webapp`. В дальнейшем Maven будет копировать эту структуру в подкаталог каталога `target`, дополняя её необходимыми файлами.

Обязательными частями этого каталога должны быть следующие:

`src\main\webapp\WEB-INF` – подкаталог, который является служебным защищаемым web-сервером от прямого доступа (то есть клиентом через браузер не могут быть скачаны никакие файлы из этого каталога).

`src\main\webapp\WEB-INF\web.xml` – дескриптор развёртывания проекта, или файл с основными настройками проекта. Его синтаксис и пример оформления будет рассмотрен ниже.

Также при копировании этого каталога в каталог target, Maven добавляет следующие обязательные подкаталоги:

`WEB-INF\classes` – подкаталог для хранения всех скомпилированных class-файлов приложения, которые автоматически запускаются сервером при развёртывании проекта.

`WEB-INF\lib` – подкаталог для хранения сторонних библиотек (JAR-файлов), которые указаны в `pom.xml` в качестве зависимостей.

1.5. Запуск простейшего приложения

Добавим в наше приложение следующий класс для хранения списка объектов класса `MyObject`:

```
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class Storage {
    private static Map<Integer, MyObject> objects = new HashMap<>();
    static {
        MyObject object;
        object = new MyObject();
        object.setFieldA("aaa");
        object.setFieldB(12.3);
        object.setFieldC(true);
        create(object);
        object = new MyObject();
        object.setFieldA("666");
        object.setFieldB(4.56);
        object.setFieldC(false);
        create(object);
        object = new MyObject();
        object.setFieldA("ввв");
        object.setFieldB(78.9);
        object.setFieldC(true);
        create(object);
    }

    public static Collection<MyObject> readAll() {
        return objects.values();
    }

    public static MyObject readById(Integer id) {
        return objects.get(id);
    }

    public static void create(MyObject object) {
        /* минимальное значение идентификатора */
        Integer id = 1;
        /* множество идентификаторов всех объектов в списке */
        Set<Integer> ids = objects.keySet();
        if(!ids.isEmpty()) {
            /* вычисление идентификатора, на 1 большего
             * максимального из существующего */
            id += Collections.max(ids);
```

```
        }
        object.setId(id);
        objects.put(id, object);
    }

    public static void update(MyObject object) {
        objects.put(object.getId(), object);
    }

    public static void delete(Integer id) {
        objects.remove(id);
    }
}
```

Для отображения списка объектов создадим класс-сервлет. Но этот класс должен наследоваться от класса `javax.servlet.http.HttpServlet`. Этот класс отсутствует в поставке Java SE (Standard Edition). Для работы с ним можно установить версию Java EE (Enterprise Edition). Однако, при развёртывании сервлета в контейнере Tomcat, все классы и интерфейсы, относящиеся к Servlets API, уже присутствуют в библиотеках Tomcat. Фактически, нам эти классы нужны только чтобы IDE могла скомпилировать наши классы. Но для этого достаточно указать необходимую библиотеку в качестве зависимости Maven. Добавим в `pom.xml` раздел `<dependencies>` с нужной зависимостью. Файл `pom.xml` тогда будет выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>by.vsu</groupId>
    <artifactId>web-application-example</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
       <maven.compiler.source>16</maven.compiler.source>
       <maven.compiler.target>16</maven.compiler.target>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

<dependencies>
    <dependency>
       <groupId>javax.servlet</groupId>
       <artifactId>javax.servlet-api</artifactId>
       <version>4.0.1</version>
       <scope>provided</scope>
    </dependency>
</dependencies>


    <build>
```

```
        <plugins>
           <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-war-plugin</artifactId>
              <version>3.3.2</version>
           </plugin>
        </plugins>
    </build>
</project>
```

Здесь **<scope>**provided**</scope>** означает, что библиотека нужна для компиляции проекта, но для запуска проекта в среде исполнения эта библиотека уже не нужна, так как она поставляется самой средой (в нашем случае контейнером сервлетов Tomcat). Поэтому эту библиотеку Maven не помещает в папку `WEB-INF/lib`.

Теперь можем описать класс-сервлет следующим образом:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        Collection<MyObject> objects = Storage.readAll();
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("<STYLE>");
        w.println("TABLE {");
        w.println("border-collapse: collapse;");
        w.println("}");
        w.println("TH, TD {");
        w.println("border: 1px solid black;");
        w.println("padding: 5px 30px 5px 10px;");
        w.println("}");
        w.println("</STYLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<TABLE>");
        w.println("<TR><TH>Поле A</TH><TH>Поле B</TH><TH>Поле C</TH></TR>");
        for(MyObject object : objects) {
            w.printf("<TR><TD>%s</TD><TD>%.2f</TD><TD>%s</TD></TR>",
                     object.getFieldA(), object.getFieldB(),
                     object.getFieldC() ? "Да" : "Нет");
        }
        w.println("</TABLE>");
```

```
        w.println("</BODY>");
        w.println("</HTML>");
    }
}
```

Дескриптор развёртывания приложения опишем следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>Тестовый пример</display-name>
    <welcome-file-list>
        <welcom-file>index.html</welcom-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>ListServlet</servlet-name>
        <servlet-class>ListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListServlet</servlet-name>
        <url-pattern>/index.html</url-pattern>
    </servlet-mapping>
</web-app>
```

В данном примере созданный сервлет `ListServlet` не реализует, как было сказано выше, интерфейс `javax.servlet.Servlet` напрямую, а наследуется от класса `javax.servlet.http.HttpServlet`, что является обычной практикой, так как избавляет программиста от ручной обработки HTTP-запросов. Для обработки запроса Tomcat автоматически будет вызывать метод `doGet()` описанного класса. Формирование HTML-ответа клиенту происходит с помощью второго параметра метода `doGet()` – объекта класса, реализующего интерфейс `javax.servlet.http.HttpServletResponse`. Этот объект позволяет с помощью метода `getWriter()` получить объект символьного выходного потокового класса, позволяющего отправлять необходимые данные клиенту.

В дескрипторе развёртывания регистрируется класс-сервлет, и создаётся для него так называемый маппинг, т.е. привязка URL-path к имени этого класса, что позволяет контейнеру сервлета, получая HTTP-запрос с указанным URL-path определять обработчик для поступившего запроса.

После создания необходимых классов и файлов, необходимо:

1. Выполнить команду Maven `package`.
2. Открыть панель управления Tomcat в браузере и нажать кнопку «Reload» в строке с приложением (можно ориентироваться либо на путь `/web-application-example`, либо на название приложения «Тестовый пример»).
3. Открыть приложение в браузере по следующему URL: `http://localhost:8080/web-application-example`

1.6. Лабораторная работа №1 «Servlets API»

Разработать набор сервлетов, позволяющих просматривать список некоторых объектов (см. вариант). Формулировка заданий по вариантам предполагает дальнейшую доработку приложения в последующих лабораторных работах.

Вариант 1

Список книг различных авторов (шифр книги, автор, название, год издания, количество страниц, количество ссылок на книгу, индекс цитируемости книги). Последнее поле должно рассчитываться автоматически как отношение количества ссылок на книгу к количеству страниц книги. В отдельное текстовое поле выводить автора (или список авторов через запятую, если таких авторов несколько) с самым высоким индексом цитируемости (суммарный индекс цитируемости всех его книг).

Вариант 2

Список заданий для различных проектов (шифр задания, название проекта, описание задания, планируемое время выполнения в часах, фактическое время выполнения в часах, отставание). Последнее поле должно рассчитываться автоматически как разность фактического и планируемого времени выполнения (но только для тех заданий, для которых отставание больше 0). Отдельно выводить название задания с наибольшим отставанием (или список названий через запятую, если таких заданий несколько).

Вариант 3

Список студентов факультета (номер зачетки, группа, фамилия, имя, отчество, оценка по 1-му предмету в сессию, оценка по 2-му предмету в сессию, оценка по 3-му предмету в сессию, оценка по 4-му предмету в сессию, средний балл). Последнее поле должно рассчитываться автоматически. В отдельное текстовое поле выводить номер группы (или список номеров групп через запятую, если таких групп несколько) средний балл студентов по которым максимален.

Вариант 4

Каталог музыкальных произведений (идентификатор в каталоге, автор (группа), название, время звучания (в формате «mm:ss»), дата публикации в каталоге, количество загрузок, популярность). Последнее поле должно рассчитываться автоматически как отношение количества загрузок к количеству дней, прошедших с момента публикации в каталоге. В отдельное текстовое поле вывести общее время звучания всех произведений, добавленных в каталог за последний месяц (вывод в формате «dd дней, hh:mm:ss»).

Вариант 5

График отпусков сотрудников организации (код сотрудника, отдел, фамилия, имя, отчество, дата начала отпуска, дата окончания отпуска, заработная плата, размер отпускных). Последнее поле должно рассчитываться автоматически исходя из того, что за каждый день отпуска любой сотрудник получает определенный процент (единый для всех сотрудников) от своей зарплаты. Также подсчитать в отдельном текстовом поле общую сумму отпускных по организации. В отдельное многострочное поле по каждому отделу привести промежуток времени, в который в отделе будет работать наименьшее количество человек.

Вариант 6

Список комнат общежития (номер комнаты, этаж, пол проживающих, количество мест, количество занятых мест, количество свободных мест). Последнее поле должно рассчитываться автоматически как разность общего количества мест и количества занятых мест. Также подсчитать в отдельных трёх текстовых полях общее число свободных мест (всего и по мужским и женским комнатам в отдельности).

Вариант 7

Список дисциплины кафедры (название дисциплины, фамилия, имя, отчество преподавателя, номер группы, количество студентов, количество лекционных часов, количество часов практических занятий, количество часов лабораторных работ, есть ли по дисциплине экзамен, есть ли по дисциплине зачёт, общее число часов). Последнее поле должно рассчитываться автоматически как сумма часов на лекционных, практических и лабораторных занятиях, и ещё к ней необходимо прибавить количество часов на экзамен (если он есть исходя из 0,5 часа на одного студента) и на зачёт (если он есть исходя из 0,25 часа на одного студента). Также подсчитать в отдельных четырёх текстовых полях общее число аудиторных часов по кафедре (сумма лекционных часов, часов практических занятий, часов лабораторных работ, и общую сумму этих трёх полей).

Вариант 8

Список филиалов торговой компании (название филиала, дата, численность торговых представителей на эту дату, объём продаж на эту дату, средний объём продаж на одного торгового представителя). Последнее поле должно рассчитываться автоматически – общий объём продаж на соответствующую дату, разделённый на численность торговых представителей на ту же дату. Также вывести в отдельном поле название филиала (или список названий филиалов через запятую, если таких несколько), которые за последнюю неделю имели минимальный из средних объёмов продаж на одного торгового представителя.

Вариант 9

Список абитуриентов, поступающих на некоторую специальность в вуз (шифр абитуриента, фамилия, имя, отчество абитуриента, средний бал аттестата, оценка по первому экзамену, оценка по второму экзамену, оценка по третьему экзамену, средний бал абитуриента). Последнее поле должно рассчитываться автоматически как среднее арифметическое среднего бала аттестата и трёх экзаменационных оценок. Также в отдельное текстовое поле вывести шифры абитуриентов (через запятую), прошедших по конкурсу на эту специальность. Для чего в специальном текстовом поле пользователь вводит количество мест на специальность, после чего проводится конкурс среди абитуриентов, имеющих по всем трём экзаменам оценку не ниже 4. Абитуриенты, участвующие в конкурсе, сортируются по общему среднему балу, при совпадении которого – по оценке за первый экзамен, после – за второй, третий и в итоге по среднему балу аттестата. После сортировки абитуриенты зачисляются по порядку убывания среднего бала до исчерпания количества свободных мест.

Вариант 10

Список правонарушений водителей (номер автотранспортного средства, фамилия, имя, отчество водителя, состав правонарушения, дата правонарушения, сумма штрафа, дата оплаты штрафа, пеня). Последнее поле должно рассчитываться автоматически в зависимости от даты оплаты. Если дата не указана, то штраф ещё не оплачен и в качестве даты оплаты при расчёте пени считается текущая дата. Если с момента правонарушения до даты оплаты прошло менее 1 месяца, пеня не начисляется (равна 0), иначе начисляется 1% от суммы штрафа за каждый день, прошедший после истечения месячного срока. Также вывести в отдельных полях общую сумму, полученную от всех штрафов, и общую сумму ещё не взысканных штрафов (сумма по штрафам и пене, но отдельно для тех правонарушений, у которых указана дата оплата штрафа, то есть оплата была произведена, и для которых эта дата не указана).

Вариант 11

Список звонков абонентов некоторого телефонного оператора (номер вызывающего абонента, номер вызываемого абонента, дата и время начала звонка, длительность звонка в секундах, стоимость одной тарифной единицы, единица тарификации в секундах, стоимость звонка). Последнее поле должно рассчитываться автоматически. Сначала необходимо вычислить длительность звонка в единицах тарификации. Например, если единица тарификации – 12 секунд, а длительность звонка – 27 секунд, то длительность звонка в единицах тарификации – 3, что соответствует 36 секундам. Далее количество единиц тарификации умножается на стоимость одной тарифной единицы. Также вывести в отдельном поле номер абонента (или список таких номеров через запятую, если из несколько), имеющих самую большую суммарную длительность звонков (как входящих, так и исходящих).

Вариант 12

Список путёвок некоторого туроператора (страна, город, количество дней, количество человек, стоимость проезда на одного человека, стоимость 1 дня проживания на одного человека, стоимость питания за день на одного человека, общая стоимость путёвки). Последнее поле должно рассчитываться автоматически, с учётом скидки 10% от общей суммы при количестве человек свыше 3. Также вывести в отдельном поле название страны (или список таких стран через запятую, если их несколько), количество путёвок в которую максимально.

Вариант 13

Список талонов к врачам поликлиники на текущий день (специальность врача, фамилия и инициалы врача, фамилия и инициалы пациента, номер очереди, время начала приёма, время окончания приёма). Последнее два поля должны рассчитываться автоматически исходя из времени работы специалиста с 800 до 1400 и времени, отводимом на приём одного пациента – 15 мин. Также вывести в отдельном поле фамилию пациента (или список таких пациентов через запятую, если их несколько), у которых имеется наибольшее количество талонов за день.

Вариант 14

Список платежей, проводимых в банке за день (время перевода, номер счёта-отправителя, номер счёта получателя, сумма для перечисления, комиссия). Последнее поле должно рассчитываться автоматически как 3% от суммы перечисления, при этом на счёт получателя перечисляется вся сумма, а со счёта отправителя считывается 103% от суммы. Также вывести в отдельные поля номера счетов (или список таких номеров через запятую, если их несколько), для которых дневной баланс перечислений является максимальным (в первое поле) и минимальным (во второе поле).

Вариант 15

Список вещей в ломбарде (название вещи, дата поступления, статус (храниться, выкуплена, продана), оценочная стоимость, выкупная стоимость). Последнее поле должно автоматически вычисляться. При статусе «храниться» – как 50% оценочной стоимости плюс 1% от оценочной стоимости за каждый день хранения. При статусе «выкуплена» или «продана» выкупная стоимость не вычисляется (равна 0). Также вывести в отдельном поле название (или список таких названий через запятую, если их несколько) самой «выгодной» вещи, для которой разница между выкупной стоимостью и оценочной стоимостью максимальна.

2. ОБРАБОТКА HTTP-ЗАПРОСОВ

2.1. Обработка данных форм

Реализуем возможность добавления нового объекта. Для этого в каталоге `src\main\webapp` создадим статичный HTML-файл с формой для ввода данных о новом объекте:

```
<HTML>
<HEAD>
<META http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
<TITLE>Тест</TITLE>
</HEAD>
<BODY>
<FORM action="save.html">
<P>Поле A:</P>
<INPUT type="text" name="field-a">
<P>Поле B:</P>
<INPUT type="text" name="field-b">
<P><INPUT type="checkbox" name="field-c" value="value"> Поле C.</P>
<BUTTON type="submit">Сохранить</BUTTON> <A href="index.html">Назад</A>
</FORM>
</BODY>
</HTML>
```

В сервлете, генерирующем список объектов, добавим ссылку на форму добавления объекта:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        Collection<MyObject> objects = Storage.readAll();
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("<STYLE>");
        w.println("TABLE {");
        w.println("border-collapse: collapse;");
        w.println("}");
        w.println("TH, TD {");
        w.println("border: 1px solid black;");
```

```
        w.println("padding: 5px 30px 5px 10px;");
        w.println("}");
        w.println("</STYLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<TABLE>");
        w.println("<TR><TH>Поле A</TH><TH>Поле B</TH><TH>Поле C</TH></TR>");
        for(MyObject object : objects) {
            w.printf("<TR><TD>%s</TD><TD>%.2f</TD><TD>%s</TD></TR>",
                     object.getFieldA(), object.getFieldB(),
                     object.getFieldC() ? "Да" : "Нет");
        }
        w.println("</TABLE>");
        w.println("<P><A href=\"edit.html\">Добавить</A></P>");
        w.println("</BODY>");
        w.println("</HTML>");
    }
}
```

Для обработки данных формы создадим отдельный класс `SaveServlet`. Данный сервлет не будет сам генерировать HTML-страницу браузеру. После обработки формы логичным действием будет снова отобразить пользователю список объектов. Но для отображения списка объектов можно после обработки данных формы передать управление классу `ListServlet`:

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SaveServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        MyObject object = new MyObject();
        object.setFieldA(req.getParameter("field-a"));
        object.setFieldB(Double.parseDouble(req.getParameter("field-b")));
        object.setFieldC(req.getParameter("field-c") != null);
        Storage.create(object);
        /* перенаправление запроса на дальнейшую обработку
         * другому сервлету */
        getServletContext().getRequestDispatcher("/index.html")
                                                    .forward(req, resp);
    }
}
```

Модифицируем также дескриптор развёртывания:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>Тестовый пример</display-name>
    <welcome-file-list>
        <welcom-file>index.html</welcom-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>ListServlet</servlet-name>
        <servlet-class>ListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListServlet</servlet-name>
        <url-pattern>/index.html</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>SaveServlet</servlet-name>
        <servlet-class>SaveServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SaveServlet</servlet-name>
        <url-pattern>/save.html</url-pattern>
    </servlet-mapping>

</web-app>
```

Как видно из примера, в сервлете `SaveServlet` для обработки данных формы применяется метод `getParameter()` класса `HttpServletRequest`. При этом любые элементы управления на HTML-форме, будь то обычный элемент для ввода текста с клавиатуры (`type="text"`), независимый (`type="checkbox"`) или зависимый (`type="radio"`) переключатели, многострочное поле для ввода текста (элемент `<textarea>`), выпадающий список (элемент `<select>`) и др., все эти данные приходят на сервер в виде пар `<имя>=<значение>`. После чего по имени можно с помощью метода `getParameter()` получить необходимое (определённое пользователем) значение.

Важно! *Если для какого-либо элемента HTML-формы не задан атрибут `name (<имя>)`, то значение этого элемента на сервер не передаётся.*

Также необходимо обратить внимание на следующую сроку в классе `SaveServlet`:

```
getServletContext().getRequestDispatcher("/index.html")
                                        .forward(req, resp);
```

Таким способом можно передавать запрос на обработку от одного сервелета другому. Но важно помнить три особенности этого механизма:

1. Передача управления происходит в рамках обработки запроса на сервере. То есть такая передача управления остаётся скрытой от клиента. Для клиента вся обработка запроса – это некое единое

действие, результат которого он получает в конечном итоге. А способ, которым результат был получен, клиенту остаётся не известен.

2. Метод `forward()` вызывает у второго сервлета тот же метод по обработке запроса, что и у текущего сервлета. То есть если у текущего сервлета вызывался метод `doGet()`, то и у вызываемого сервлета выполняется `doGet()`. Именно поэтому у класса `SaveServlet` на данном шаге примера описан метод `doGet()`, а не `doPost()`, как было бы логичнее для обработки данных формы. Но в таком случае передать управление сервлету `ListServlet` уже было бы невозможно, так как у него нет метода `doPost()`. Эта проблема будет решаться на следующем шаге.
3. При передаче управления другому сервлету, мы не указываем имя класса для этого сервлета, а лишь URL-path, по которому контейнер сервлетов может его найти, используя маппинги из дескриптора развёртывания. Но такой механизм позволяет передавать управление не только сервлетам, но и JSP-страницам.

Помимо безвозвратной передачи управления методом `forward()`, есть возможность временно (с возвратом) передавать управление методом `include()`.

2.2. Операция redirect

Текущая версия приложения имеет один существенный недостаток. Когда форма с данными о новом объекте отправляется на сервер по адресу `/save.html`, на сервере класс `SaveServlet` сохраняет новый объект, а затем передаёт управление классу `ListServlet`, который формирует новый список объектов. Но с точки зрения браузера полученный список — это результат обращения по адресу `/save.html`. Всё это приводит к тому, что обновление страницы после нажатия кнопки «Сохранить» заставляет браузер продублировать отправку формы и, следственно, объект в списке.

Кроме описанного недостатка есть ещё одно неудобство, же описанное выше. Обработка данных формы в классе `SaveServlet` осуществляется методом `doGet()`, хотя в данном случае правильным было бы использование метода `doPost()`. Но если перенаправлять запрос из метода `doPost()` класса `SaveServlet` в класс `ListServlet`, то это также приводит к вызову метода `doPost()` класса `ListServlet`, который мы не определяли.

Общим решением обоих проблем является использование перенаправления запроса от одного сервлета другому через отправку соответствующего ответа браузеру. То есть вместо явного перенаправления запроса классу `ListServlet`, класс `SaveServlet` отправляет ответ браузеру с указанием перейти на страницу `/index.html`. После этого

ответа браузер автоматически отправляет новый GET-запрос (при чём всегда только GET-запрос) на указанную страницу, который будет обработан классом `ListServlet`, но в этом случае это будет явное обращение по адресу `/index.html`, что уже не приводит к повторной отправке данных на сервер в случае обновления страницы. Также в этом случае мы без труда можем использовать метод `doPost()` в классе `SaveServlet` продолжая использовать метод `doGet()` в классе `ListServlet`.

Изменённый `SaveServlet.java`

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SaveServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        MyObject object = new MyObject();
        object.setFieldA(req.getParameter("field-a"));
        object.setFieldB(Double.parseDouble(req.getParameter("field-b")));
        object.setFieldC(req.getParameter("field-c") != null);
        Storage.create(object);

        /* отправка ответа в браузер с указанием перейти по другому URL */
        resp.sendRedirect(req.getContextPath() + "/index.html");

    }
}
```

Изменённый `edit.html`

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<TITLE>Тест</TITLE>
</HEAD>
<BODY>
<FORM action="save.html" method="post">
<P>Поле A:</P>
<INPUT type="text" name="field-a">
<P>Поле B:</P>
<INPUT type="text" name="field-b">
<P><INPUT type="checkbox" name="field-c" value="value"> Поле C.</P>
<BUTTON type="submit">Сохранить</BUTTON> <A href="index.html">Назад</A>
</FORM>
</BODY>
</HTML>
```

Описанный механизм, реализуемый с помощью метода `sendRedirect()` класса `HttpServletResponse`, и обозначается термином redirect.

2.3. Динамическая генерация форм

Реализуем теперь возможность редактирования уже существующего объекта. Форма редактирования, по сути, является точно такой же, что и форма создания нового объекта, только она должна быть заполнена данными, соответствующему выбранному пользователем объекту. Так как форма будет динамически заполняться данными, то она должна генерироваться сервлетом. Создадим класс `EditServlet` (вместо статичной страницы `/edit.html`, которую мы удалим), генерирующий форму редактирования (или оставляющий её пустой, если необходимо добавить новый объект). Для того, чтобы данный класс «знал», данные о каком объекте необходимо подставить в форму, этому сервлету методом GET можно передать уникальный идентификатор объекта. Если же такой идентификатор не передаётся, это означает, что генерируется форма создания нового объекта. Но при этом обработчик формы (класс `SaveServlet`) тоже должен знать, нужно ли сохранять новый объект или обновлять информацию о существующем. Для этого ему тоже необходимо передавать (или не передавать) идентификатор объекта. Так как на страницу редактирования объекта пользователь может попасть с главной страницы, выбрав соответствующий объект из списка, то идентификатор выбранного объекта можно подставлять как GET-параметр непосредственно в тег гиперссылки `A`. Классу `SaveServlet` все данные передаются с формы, поэтому и идентификатор можно передавать на форме, используя для этого скрытый (`type="hidden"`) элемент формы.

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EditServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        MyObject object = null;
        try {
            object = Storage.readById(Integer.parseInt(
                                                req.getParameter("id")));
        } catch(NumberFormatException e) {}
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
```

```
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<FORM action=\"save.html\" method=\"post\">");
        if(object != null) {
            w.printf("<INPUT type=\"hidden\" name=\"id\" value=\"%s\">\n",
                                              object.getId().toString());
        }
        w.println("<P>Поле A:</P>");
        w.printf("<INPUT type=\"text\" name=\"field-a\" value=\"%s\">\n",
                     object != null ? object.getFieldA() : new String());
        w.println("<P>Поле B:</P>");
        w.printf("<INPUT type=\"text\" name=\"field-b\" value=\"%s\">\n",
           object != null ? object.getFieldB().toString() : new String());
        w.println("<P><INPUT type=\"checkbox\" name=\"field-c\""
              + " value=\"value\"" + (object != null && object.getFieldC()
                          ? " checked" : new String()) + "> Поле C.</P>");
        w.println("<BUTTON type=\"submit\">Сохранить</BUTTON>");
        w.println("<A href=\"index.html\">Назад</A>");
        w.println("</FORM>");
        w.println("</BODY>");
        w.println("</HTML>");
    }
}
```

Внесём соответствующие изменения в дескриптор развёртывания:
Изменённый `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>Тестовый пример</display-name>
    <welcome-file-list>
        <welcom-file>index.html</welcom-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>ListServlet</servlet-name>
        <servlet-class>ListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListServlet</servlet-name>
        <url-pattern>/index.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>EditServlet</servlet-name>
        <servlet-class>EditServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>EditServlet</servlet-name>
        <url-pattern>/edit.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>SaveServlet</servlet-name>
```

```
        <servlet-class>SaveServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SaveServlet</servlet-name>
        <url-pattern>/save.html</url-pattern>
    </servlet-mapping>
</web-app>
```

Добавим ссылки на объекты на главной странице:
Изменённый `ListServlet.java`

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        Collection<MyObject> objects = Storage.readAll();
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("<STYLE>");
        w.println("TABLE {");
        w.println("border-collapse: collapse;");
        w.println("}");
        w.println("TH, TD {");
        w.println("border: 1px solid black;");
        w.println("padding: 5px 30px 5px 10px;");
        w.println("}");
        w.println("</STYLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<TABLE>");
        w.println("<TR><TH>Поле A</TH><TH>Поле B</TH><TH>Поле C</TH></TR>");
        for(MyObject object : objects) {
          w.print("<TR>");
          w.printf("<TD><A href=\"edit.html?id=%d\">%s</A></TD>",
                   object.getId(), object.getFieldA());
          w.printf("<TD>%.2f</TD>", object.getFieldB());
          w.printf("<TD>%s</TD>", object.getFieldC() ? "Да" : "Нет");
          w.println("</TR>");
        }
        w.println("</TABLE>");
        w.println("<P><A href=\"edit.html\">Добавить</A></P>");
        w.println("</BODY>");
```

```
        w.println("</HTML>");
    }
}
```

Внесём соответственные изменения в обработчик формы:

Изменённый `SaveServlet.java`

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SaveServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        MyObject object = new MyObject();
        object.setFieldA(req.getParameter("field-a"));
        object.setFieldB(Double.parseDouble(req.getParameter("field-b")));
        object.setFieldC(req.getParameter("field-c") != null);

        try {
            object.setId(Integer.parseInt(req.getParameter("id")));
        } catch(NumberFormatException e) {}
        if(object.getId() == null) {
            Storage.create(object);
        } else {
            Storage.update(object);
        }

        resp.sendRedirect(req.getContextPath() + "/index.html");
    }
}
```

2.4. Обработка одноимённых параметров формы

Добавим теперь возможность удалять объекты в приложении. Для этого на главной странице в списке объектов добавим ещё одну колонку в таблице, в которой рядом с каждым объектом поместим флажок-переключатель для отметки удаляемых объектов, а внизу таблицы добавим кнопку «Удалить». Форму со всеми флажками-переключателями будет отправлять по адресу `/delete.html`:

Изменённый `ListServlet.java`

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        Collection<MyObject> objects = Storage.readAll();
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("<STYLE>");
        w.println("TABLE {");
        w.println("border-collapse: collapse;");
        w.println("}");
        w.println("TH, TD {");
        w.println("border: 1px solid black;");
        w.println("padding: 5px 30px 5px 10px;");
        w.println("}");
        w.println("</STYLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<FORM action=\"delete.html\" method=\"post\">");
        w.println("<TABLE>");
        w.print("<TR>");
        w.print("<TH> </TH>");
        w.print("<TH>Поле A</TH>");
        w.print("<TH>Поле B</TH>");
        w.print("<TH>Поле C</TH>");
        w.println("</TR>");
        for(MyObject object : objects) {
            w.print("<TR>");
            w.printf("<TD>");
            w.printf("<INPUT type=\"checkbox\" name=\"id\" value=\"%d\">",
                    object.getId());
            w.printf("</TD>");
            w.printf("<TD><A href=\"edit.html?id=%d\">%s</A></TD>",
                    object.getId(), object.getFieldA());
            w.printf("<TD>%.2f</TD>", object.getFieldB());
            w.printf("<TD>%s</TD>", object.getFieldC() ? "Да" : "Нет");
            w.println("</TR>");
        }
        w.println("</TABLE>");
        w.println("<P>");
        w.println("<A href=\"edit.html\">Добавить</A>");
        w.println("<BUTTON type=\"submit\">Удалить</BUTTON>");
        w.println("</P>");
        w.println("</FORM>");
```

```
        w.println("</BODY>");
        w.println("</HTML>");
    }
}
```

Для адреса `/delete.html` назначим в дескрипторе развёртывания обработчик — класс `DeleteServlet`:

Изменённый `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>Тестовый пример</display-name>
    <welcome-file-list>
        <welcom-file>index.html</welcom-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>ListServlet</servlet-name>
        <servlet-class>ListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListServlet</servlet-name>
        <url-pattern>/index.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>EditServlet</servlet-name>
        <servlet-class>EditServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>EditServlet</servlet-name>
        <url-pattern>/edit.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>SaveServlet</servlet-name>
        <servlet-class>SaveServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SaveServlet</servlet-name>
        <url-pattern>/save.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>DeleteServlet</servlet-name>
        <servlet-class>DeleteServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DeleteServlet</servlet-name>
        <url-pattern>/delete.html</url-pattern>
    </servlet-mapping>

</web-app>
```

Сам класс `DeleteServlet` реализуем по аналогии с классом `SaveServlet`, только для получения данных с формы нам для атрибута `id` нужно получить список значений идентификаторов, соответствующих

выбранным пользователям объектам. Для этого воспользуемся методом `getParameterValues()` класса `HttpServletRequest`:

DeleteServlet.java

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DeleteServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        for(String id : req.getParameterValues("id")) {
            try {
                Storage.delete(Integer.parseInt(id));
            } catch(NumberFormatException e) {}
        }
        resp.sendRedirect(req.getContextPath() + "/index.html");
    }
}
```

2.5. Использование операции forward для разделения ответственности сервлетов по обработке запросов. Использование атрибутов запроса для передачи данных между сервлетами

Функционал примера для практически реализован. Однако, следует произвести некоторые изменения в структуре исходного кода. Дело в том, что классы `ListServlet` и `EditServlet` в данной версии занимаются подготовкой данных и генерацией HTML-страницы на основе подготовленных данных. В данном простом примере подготовка данных не сложная и не загромождает сервлет. Но в более сложных задачах, в которых и обработка данных может быть сложнее (то есть логика функционирования приложения), и может добавляться задача проверки корректности данных, работа с сессиями пользователей и контроль доступа пользователя к данным и т. д., то исходный код такого сервлета становится слишком объёмным и громоздким. Поэтому принято разделять задачу подготовки данных и задачу отображения данных (генерацию HTML-страниц на их основе) на два различных сервлета. При этом первый сервлет после подготовки данных перенаправляет запрос на второй сервлет, который эти данные отображает. Учитывая, что перенаправление запроса от сервлета к сервлету не предполагает прямого взаимодействия соответствующих классов (см. пример классов `SaveServlet` и `ListServlet` в п. 2.1), то для передачи данных будем использовать атрибуты запроса. Также сервлеты, которые отвечают за отображение данных, будут назначаться как обработчики

адресов, начинающихся с папки WEB-INF, что защищает их от прямого обращения из браузера. В противном случае сервлеты не получали бы никаких данных из атрибутов запроса, что может приводить к нестабильной работе приложения. Таким образом, получаем следующие изменения:

Изменённый web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>Тестовый пример</display-name>
    <welcome-file-list>
        <welcom-file>index.html</welcom-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>ListServlet</servlet-name>
        <servlet-class>ListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListServlet</servlet-name>
        <url-pattern>/index.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>ViewListServlet</servlet-name>
        <servlet-class>ViewListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ViewListServlet</servlet-name>
        <url-pattern>/WEB-INF/index.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>EditServlet</servlet-name>
        <servlet-class>EditServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>EditServlet</servlet-name>
        <url-pattern>/edit.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>ViewEditServlet</servlet-name>
        <servlet-class>ViewEditServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ViewEditServlet</servlet-name>
        <url-pattern>/WEB-INF/edit.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>SaveServlet</servlet-name>
        <servlet-class>SaveServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SaveServlet</servlet-name>
        <url-pattern>/save.html</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>DeleteServlet</servlet-name>
```

```
        <servlet-class>DeleteServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DeleteServlet</servlet-name>
        <url-pattern>/delete.html</url-pattern>
    </servlet-mapping>
</web-app>
```

Изменённый ListServlet.java

```
import java.io.IOException;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        Collection<MyObject> objects = Storage.readAll();
        req.setAttribute("objects", objects);
        getServletContext().getRequestDispatcher("/WEB-INF/index.html")
                                                        .forward(req, resp);
    }
}
```

ViewListServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ViewListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        @SuppressWarnings("unchecked")
        Collection<MyObject> objects = (Collection<MyObject>)req
                                                .getAttribute("objects");
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("<STYLE>");
        w.println("TABLE {");
        w.println("border-collapse: collapse;");
        w.println("}");
        w.println("TH, TD {");
```

```
        w.println("border: 1px solid black;");
        w.println("padding: 5px 30px 5px 10px;");
        w.println("}");
        w.println("</STYLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<FORM action=\"delete.html\" method=\"post\">");
        w.println("<TABLE>");
        w.print("<TR>");
        w.print("<TH> </TH>");
        w.print("<TH>Поле A</TH>");
        w.print("<TH>Поле B</TH>");
        w.print("<TH>Поле C</TH>");
        w.println("</TR>");
        for(MyObject object : objects) {
            w.print("<TR>");
            w.printf("<TD>");
            w.printf("<INPUT type=\"checkbox\" name=\"id\" value=\"%d\">",
                     object.getId());
            w.printf("</TD>");
            w.printf("<TD><A href=\"edit.html?id=%d\">%s</A></TD>",
                     object.getId(), object.getFieldA());
            w.printf("<TD>%.2f</TD>", object.getFieldB());
            w.printf("<TD>%s</TD>", object.getFieldC() ? "Да" : "Нет");
            w.println("</TR>");
        }
        w.println("</TABLE>");
        w.println("<P>");
        w.println("<A href=\"edit.html\">Добавить</A>");
        w.println("<BUTTON type=\"submit\">Удалить</BUTTON>");
        w.println("</P>");
        w.println("</FORM>");
        w.println("</BODY>");
        w.println("</HTML>");
    }
}
```

Изменённый `EditServlet.java`

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EditServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        try {
            Integer id = Integer.parseInt(req.getParameter("id"));
            MyObject object = Storage.readById(id);
            req.setAttribute("object", object);
        } catch(NumberFormatException e) {}
        getServletContext().getRequestDispatcher("/WEB-INF/edit.html")
                                                        .forward(req, resp);
    }
}
```

ViewEditServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ViewEditServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        MyObject object = (MyObject)req.getAttribute("object");
        resp.setCharacterEncoding("UTF-8");
        PrintWriter w = resp.getWriter();
        w.println("<HTML>");
        w.println("<HEAD>");
        w.println("<META http-equiv=\"Content-Type\"");
        w.println("      content=\"text/html; charset=UTF-8\">");
        w.println("<TITLE>Тест</TITLE>");
        w.println("</HEAD>");
        w.println("<BODY>");
        w.println("<FORM action=\"save.html\" method=\"post\">");
        if(object != null) {
            w.printf("<INPUT type=\"hidden\" name=\"id\" value=\"%d\">\n",
                                              object.getId());
        }
        w.println("<P>Поле A:</P>");
        w.printf("<INPUT type=\"text\" name=\"field-a\" value=\"%s\">\n",
                     object != null ? object.getFieldA() : new String());
        w.println("<P>Поле B:</P>");
        w.printf("<INPUT type=\"text\" name=\"field-b\" value=\"%s\">\n",
           object != null ? object.getFieldB().toString() : new String());
        w.println("<P><INPUT type=\"checkbox\" name=\"field-c\""
              + " value=\"value\"" + (object != null && object.getFieldC()
                           ? " checked" : new String()) + "> Поле C.</P>");
        w.println("<BUTTON type=\"submit\">Сохранить</BUTTON>");
        w.println("<A href=\"index.html\">Назад</A>");
        w.println("</FORM>");
        w.println("</BODY>");
        w.println("</HTML>");
    }
}
```

Кроме того, разделение обработки и визуализации данных позволит вместо сервлетов, отвечающих за визуализацию, в будущем использовать другие технологии, например, JSP-страницы.

2.6. Лабораторная работа №2 «Обработка HTTP-запросов»

Доработать набор сервлетов из лабораторной работы №1 так, чтобы приложение позволяло не только просматривать список объектов, но и управлять этим списком: добавлять, изменять и удалять объекты через web-интерфейс.

Предусмотреть обработку ошибочно введенных данных. Реализовать обработку данных, предусмотренных вариантом (хранение данных на сервере реализовать через сериализацию).

3. JAVA SERVER PAGES

Технология Java Server Pages (JSP) – это, по сути, технология генерации «на лету» сервлетов, отвечающих за визуализацию (генерацию HTML-страниц), а также компиляцию и запуск этих сервлетов также «на лету». При этом технология специально разработана для более удобной работы с HTML-кодом и абстрагирования программиста от Java-кода генерируемых сервлетов.

3.1. Замена View-сервлетов на JSP

Реализуем генерацию HTML-страниц с помощью JSP-страниц вместо сервлетов. Для этого удалим классы `ViewEditServlet` и `ViewListServlet`. Удаляемые классы использовали для визуализации экземпляры класса `MyObject`. Так как JSP-парсер при анализе JSP-страницы создаёт соответствующий Java-класс, который помещается в специальный пакет. Из-за того, что класс, соответствующий JSP-странице, находится в некотором пакете, и должен обращаться к классу `MyObject` (который находится в безымянном пакете и потому не может быть доступен из других пакетов), необходимо перенести класс `MyObject` в некоторый пакет, например, `pckg`.

Изменённый `MyObject.java`

```
package pckg;

public class MyObject {
    private Integer id;
    private String fieldA;
    private Double fieldB;
    private Boolean fieldC;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getFieldA() {
        return fieldA;
    }

    public void setFieldA(String fieldA) {
        this.fieldA = fieldA;
    }
```

```
    public Double getFieldB() {
        return fieldB;
    }

    public void setFieldB(Double fieldB) {
        this.fieldB = fieldB;
    }

    public Boolean getFieldC() {
        return fieldC;
    }

    public void setFieldC(Boolean fieldC) {
        this.fieldC = fieldC;
    }
}
```

Далее необходимо убрать из дескриптора развёртывания `web.xml` записи, касающиеся удалённых классов.

Изменённый `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
     <display-name>Тестовый пример</display-name>
     <welcome-file-list>
          <welcom-file>index.html</welcom-file>
     </welcome-file-list>
     <servlet>
          <servlet-name>ListServlet</servlet-name>
          <servlet-class>ListServlet</servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>ListServlet</servlet-name>
          <url-pattern>/index.html</url-pattern>
     </servlet-mapping>
     <servlet>
          <servlet-name>EditServlet</servlet-name>
          <servlet-class>EditServlet</servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>EditServlet</servlet-name>
          <url-pattern>/edit.html</url-pattern>
     </servlet-mapping>
     <servlet>
          <servlet-name>SaveServlet</servlet-name>
          <servlet-class>SaveServlet</servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>SaveServlet</servlet-name>
          <url-pattern>/save.html</url-pattern>
     </servlet-mapping>
     <servlet>
          <servlet-name>DeleteServlet</servlet-name>
          <servlet-class>DeleteServlet</servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>DeleteServlet</servlet-name>
          <url-pattern>/delete.html</url-pattern>
     </servlet-mapping>
</web-app>
```

После этого создадим в папке WEB-INF подпапку jsp. Помещать JSP-страницы внутри папки WEB-INF нужно для того, чтобы обезопасить JSP-страницы от прямого обращения. Дело в том, что JSP-страница будет получать управление от соответствующего сервлета, который должен подготовить и поместить в атрибуты запроса (HttpServletRequest) данные для JSP-страницы. А если пользователь откроет эту страницу напрямую, минуя сервлет, то JSP-страница этих данных не получит, что, в общем случае, может непредсказуемым образом отразится на функциональности, надёжности и безопасности работы JSP-страницы. Поэтому такие JSP-страницы лучше помещать в папку WEB-INF, файлы из которой не доступны для прямого доступа, но доступны для любого сервлета, который может перенаправить запрос на такую страницу (используя методы forward или include). Создание же подпапки jsp нужно всего лишь для удобства, чтобы не смешивать скомпилированные Java-классы, JSP-страницы и прочие файлы (например, сторонние библиотеки, файлы ресурсов, настроек, свойств и т. д.).

В созданной подпапке создадим два файла:

index.jsp

```
<%@page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>

<%@page import="pckg.MyObject"%>
<%@page import="java.util.Collection"%>

<%
    @SuppressWarnings("unchecked")
    Collection<MyObject> objects
                    = (Collection<MyObject>)request.getAttribute("objects");
%>

<HTML>
    <HEAD>
        <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <TITLE>Тест JSP</TITLE>
        <STYLE>
            TABLE {
                border-collapse: collapse;
            }
            TH, TD {
                border: 1px solid black;
                padding: 5px 30px 5px 10px;
            }
        </STYLE>
    </HEAD>
    <BODY>
        <FORM action="delete.html" method="post">
            <TABLE>
                <TR>
                    <TH> </TH>
                    <TH>Поле A</TH>
                    <TH>Поле B</TH>
                    <TH>Поле C</TH>
```

```
                </TR>
                <%
                    for(MyObject object : objects) {
                %>
                    <TR>
                        <TD>
                            <INPUT type="checkbox" name="id"
                                   value="<%= object.getId() %>">
                        </TD>
                        <TD>
                            <A href="edit.html?id=<%= object.getId() %>">
                                <%= object.getFieldA() %>
                            </A>
                        </TD>
                        <TD><%= object.getFieldB() %></TD>
                        <TD><%= object.getFieldC() ? "Да" : "Нет" %></TD>
                    </TR>
                <%
                    }
                %>
            </TABLE>
            <P>
                <A href="edit.html">Добавить</A>
                <BUTTON type="submit">Удалить</BUTTON>
            </P>
        </FORM>
    </BODY>
</HTML>
```

edit.jsp

```
<%@page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>

<%@page import="pckg.MyObject"%>

<%
    MyObject object = (MyObject)request.getAttribute("object");
%>
<HTML>
    <HEAD>
        <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <TITLE>Тест JSP</TITLE>
    </HEAD>
    <BODY>
        <FORM action="save.html" method="post">
            <%
                if(object != null) {
            %>
                <INPUT type="hidden" name="id"
                       value="<%= object.getId() %>">
            <%
                }
            %>
            <P>Поле A:</P>
            <INPUT type="text" name="field-a"
                   value="<%= object != null
                             ? object.getFieldA()
                             : new String() %>">
```

```
            <P>Поле B:</P>
            <INPUT type="text" name="field-b"
                   value="<%= object != null
                              ? object.getFieldB()
                              : new String() %>">
            <P><INPUT type="checkbox" name="field-c" value="value"
                      <%
                          if(object != null && object.getFieldC()) {
                      %>
                          checked
                      <%
                          }
                      %>>Поле C.</P>
            <BUTTON type="submit">Сохранить</BUTTON>
            <A href="index.html">Назад</A>
        </FORM>
    </BODY>
</HTML>
```

Также внесём соответствующие изменения в классы `ListServlet` и `EditServlet`:

Изменённый `ListServlet.java`

```
import java.io.IOException;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import pckg.MyObject;

public class ListServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        Collection<MyObject> objects = Storage.readAll();
        req.setAttribute("objects", objects);
        getServletContext().getRequestDispatcher("/WEB-INF/jsp/index.jsp")
                                                .forward(req, resp);
    }
}
```

Изменённый `EditServlet.java`

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import pckg.MyObject;

public class EditServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        try {
            Integer id = Integer.parseInt(req.getParameter("id"));
            MyObject object = Storage.readById(id);
            req.setAttribute("object", object);
        } catch(NumberFormatException e) {}
        getServletContext().getRequestDispatcher("/WEB-INF/jsp/edit.jsp")
                                                    .forward(req, resp);
    }
}
```

Для того, чтобы JSP-страницы успешно обрабатывались в IDE, необходимо добавить Maven-зависимость в `pom.xml`:

```
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
</dependency>
```

3.2. JSP Standard Tag Library

Для того, чтобы не смешивать на JSP-странице HTML-теги и исходный код на языке программирования Java, необходимо исключить скриптлеты (`<% ... %>`) и выражения (`<%= ... %>`). Вместо них будем использовать специальные JSTL-теги и Expression Language:

Для этого необходимо подключить к проекту библиотеку JSTL, добавив в `pom.xml` следующую зависимость:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

После этого в JSP-страницах можно подключать теги из этой библиотеки с помощью директивы `<%@taglib%>`.

Изменённый `index.jsp`

```
<%@page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<HTML>
```

```
    <HEAD>
        <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <TITLE>Тест JSP</TITLE>
        <STYLE>
            TABLE {
                border-collapse: collapse;
            }
            TH, TD {
                border: 1px solid black;
                padding: 5px 30px 5px 10px;
            }
        </STYLE>
    </HEAD>
    <BODY>
        <FORM action="delete.html" method="post">
            <TABLE>
                <TR>
                    <TH> </TH>
                    <TH>Поле A</TH>
                    <TH>Поле B</TH>
                    <TH>Поле C</TH>
                </TR>
                <c:forEach var="object" items="${objects}">
                    <TR>
                        <TD>
                            <INPUT type="checkbox" name="id"
                                value="${object.id}">
                        </TD>
                        <TD>
                            <A href="edit.html?id=${object.id}">
                                ${object.fieldA}
                            </A>
                        </TD>
                        <TD>${object.fieldB}</TD>
                        <TD>
                            <c:choose>
                                <c:when test="${object.fieldC}">Да</c:when>
                                <c:otherwise>Нет</c:otherwise>
                            </c:choose>
                        </TD>
                    </TR>
                </c:forEach>
            </TABLE>
            <P>
                <A href="edit.html">Добавить</A>
                <BUTTON type="submit">Удалить</BUTTON>
            </P>
        </FORM>
    </BODY>
</HTML>
```

Изменённый `edit.jsp`

```
<%@page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<c:choose>
    <c:when test="${not empty object}">
        <c:set var="fieldA" value="${object.fieldA}"/>
        <c:set var="fieldB" value="${object.fieldB}"/>
        <c:set var="fieldC" value="${object.fieldC}"/>
    </c:when>
    <c:otherwise>
        <c:set var="fieldA" value=""/>
        <c:set var="fieldB" value=""/>
        <c:set var="fieldC" value="false"/>
    </c:otherwise>
</c:choose>
<HTML>
    <HEAD>
        <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <TITLE>Тест JSP</TITLE>
    </HEAD>
    <BODY>
        <FORM action="save.html" method="post">
            <c:if test="${not empty object}">
                <INPUT type="hidden" name="id" value="${object.id}">
            </c:if>
            <P>Поле A:</P>
            <INPUT type="text" name="field-a" value="${fieldA}">
            <P>Поле B:</P>
            <INPUT type="text" name="field-b" value="${fieldB}">
            <c:if test="${fieldC}">
                <c:set var="checked" value="checked"/>
            </c:if>
            <P>
                <INPUT type="checkbox" name="field-c"
                       value="value" ${checked}>Поле C.
            </P>
            <BUTTON type="submit">Сохранить</BUTTON>
            <A href="index.html">Назад</A>
        </FORM>
   </BODY>
</HTML>
```

3.3. Лабораторная работа №3 «JSP. JSTL»

Заменить генерацию HTML-страниц в проекте на JSP-страницы и библиотеку JSTL.

4. JAVA DATABASE CONNECTIVITY

4.1. Пример взаимодействия с базой данных

Для создания структуры базы данных и наполнения таблицы тестовыми данными создадим файл с SQL-запросами (пример создан для сервера баз данных MySQL, но вопросы запуска сервера баз данных и выполнения SQL запросов из приведённого файла здесь не рассматриваются):

`init.sql`

```
DROP DATABASE IF EXISTS `my_db`;

CREATE DATABASE `my_db` DEFAULT CHARACTER SET utf8;

USE `my_db`;

CREATE TABLE `my_objects` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `field_a` VARCHAR(255) NOT NULL,
    `field_b` DOUBLE NOT NULL,
    `field_c` BOOLEAN NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARACTER SET utf8;

INSERT INTO `my_objects`
(`id`, `field_a`,     `field_b`, `field_c`)
VALUES
(1,    "aaa from DB", 12.3,      TRUE),
(2,    "666 from DB", 4.56,      FALSE),
(3,    "ввв from DB", 78.9,      TRUE);
```

Для взаимодействия с базой данных необходимо добавить в `pom.xml` зависимость для JDBC-драйвера:

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.1.0</version>
    <scope>runtime</scope>
</dependency>
```

Здесь **`<scope>`**`runtime`**`</scope>`** является противоположностью `provided` и означает, что зависимость не нужна на этапе компиляции, а нужна только на этапе разворачивания приложения.

Далее необходимо переписать класс `Storage` для использования созданной базы данных.

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collection;

import pckg.MyObject;

public class Storage {
    private static String jdbcUrl = null;
    private static String jdbcUser = null;
    private static String jdbcPassword = null;

    public static void init(String jdbcDriver, String jdbcUrl,
                            String jdbcUser,   String jdbcPassword)
                                    throws ClassNotFoundException {
        Class.forName(jdbcDriver);
        Storage.jdbcUrl = jdbcUrl;
        Storage.jdbcUser = jdbcUser;
        Storage.jdbcPassword = jdbcPassword;
    }

    public static Collection<MyObject> readAll() throws SQLException {
        String sql = "SELECT `id`, `field_a`, `field_b`, `field_c` "
                   + "FROM `my_objects`";
        Connection c = null;
        Statement s = null;
        ResultSet r = null;
        try {
            c = getConnection();
            s = c.createStatement();
            r = s.executeQuery(sql);
            Collection<MyObject> objects = new ArrayList<>();
            while(r.next()) {
                MyObject object = new MyObject();
                object.setId(r.getInt("id"));
                object.setFieldA(r.getString("field_a"));
                object.setFieldB(r.getDouble("field_b"));
                object.setFieldC(r.getBoolean("field_c"));
                objects.add(object);
            }
            return objects;
        } finally {
            try { r.close(); } catch(NullPointerException | SQLException e) {}
            try { s.close(); } catch(NullPointerException | SQLException e) {}
            try { c.close(); } catch(NullPointerException | SQLException e) {}
        }
    }

    public static MyObject readById(Integer id) throws SQLException {
        String sql = "SELECT `field_a`, `field_b`, `field_c` "
                   + "FROM `my_objects` WHERE `id` = ?";
        Connection c = null;
        PreparedStatement s = null;
        ResultSet r = null;
        try {
            c = getConnection();
            s = c.prepareStatement(sql);
            s.setInt(1, id);
            r = s.executeQuery();
            MyObject object = null;
```

```
            if(r.next()) {
                object = new MyObject();
                object.setId(id);
                object.setFieldA(r.getString("field_a"));
                object.setFieldB(r.getDouble("field_b"));
                object.setFieldC(r.getBoolean("field_c"));
            }
            return object;
        } finally {
            try { r.close(); } catch(NullPointerException | SQLException e) {}
            try { s.close(); } catch(NullPointerException | SQLException e) {}
            try { c.close(); } catch(NullPointerException | SQLException e) {}
        }
    }

    public static void create(MyObject object) throws SQLException {
        String sql = "INSERT INTO `my_objects`(`field_a`, `field_b`, `field_c`) "
                   + "VALUES (?, ?, ?)";
        Connection c = null;
        PreparedStatement s = null;
        try {
            c = getConnection();
            s = c.prepareStatement(sql);
            s.setString(1, object.getFieldA());
            s.setDouble(2, object.getFieldB());
            s.setBoolean(3, object.getFieldC());
            s.executeUpdate();
        } finally {
            try { s.close(); } catch(NullPointerException | SQLException e) {}
            try { c.close(); } catch(NullPointerException | SQLException e) {}
        }
    }

    public static void update(MyObject object) throws SQLException {
        String sql = "UPDATE `my_objects` SET "
                   + "`field_a` = ?, `field_b` = ?, `field_c` = ? "
                   + "WHERE `id` = ?";
        Connection c = null;
        PreparedStatement s = null;
        try {
            c = getConnection();
            s = c.prepareStatement(sql);
            s.setString(1, object.getFieldA());
            s.setDouble(2, object.getFieldB());
            s.setBoolean(3, object.getFieldC());
            s.setInt(4, object.getId());
            s.executeUpdate();
        } finally {
            try { s.close(); } catch(NullPointerException | SQLException e) {}
            try { c.close(); } catch(NullPointerException | SQLException e) {}
        }
    }

    public static void delete(Integer id) throws SQLException {
        String sql = "DELETE FROM `my_objects` WHERE `id` = ?";
        Connection c = null;
        PreparedStatement s = null;
        try {
            c = getConnection();
```

```
            s = c.prepareStatement(sql);
            s.setInt(1, id);
            s.executeUpdate();
        } finally {
            try { s.close(); } catch(NullPointerException | SQLException e) {}
            try { c.close(); } catch(NullPointerException | SQLException e) {}
        }
    }

    private static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(jdbcUrl, jdbcUser, jdbcPassword);
    }
}
```

Теперь необходимо внести изменения в классы-сервлеты. Во-первых, изменив инициализацию класса-хранилища. Во-вторых, добавив обработку исключительных ситуаций, генерируемых методами класса-хранилища:

Изменённый`ListServlet.java`

```
import java.io.IOException;
import java.sql.SQLException;
import java.util.Collection;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import pckg.MyObject;

public class ListServlet extends HttpServlet {
    public static final String DRIVER = "com.mysql.cj.jdbc.Driver";
    public static final String URL = "jdbc:mysql://localhost/my_db?"
                                   + "useUnicode=true&"
                                   + "characterEncoding=UTF-8";
    public static final String USER = "root";
    public static final String PASSWORD = "12345";

    @Override
    public void init() throws ServletException {
        try {
            Storage.init(DRIVER, URL, USER, PASSWORD);
        } catch(ClassNotFoundException e) {
            throw new ServletException(e);
        }
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        try {
```

```
            Collection<MyObject> objects = Storage.readAll();
            req.setAttribute("objects", objects);
            getServletContext().
                    getRequestDispatcher("/WEB-INF/jsp/index.jsp")
                                                .forward(req, resp);
        } catch(SQLException e) {
            throw new ServletException(e);
        }
    }
}
```

Изменённый EditServlet.java

```
import java.io.IOException;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import pckg.MyObject;

public class EditServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        try {
            Integer id = Integer.parseInt(req.getParameter("id"));
            MyObject object = Storage.readById(id);
            req.setAttribute("object", object);
        } catch(NumberFormatException e) {
        } catch(SQLException e) {
            throw new ServletException(e);
        }
        getServletContext().getRequestDispatcher("/WEB-INF/jsp/edit.jsp")
                                                .forward(req, resp);
    }
}
```

Изменённый SaveServlet.java

```
import java.io.IOException;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import pckg.MyObject;

public class SaveServlet extends HttpServlet {
```

```
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        MyObject object = new MyObject();
        object.setFieldA(req.getParameter("field-a"));
        object.setFieldB(Double.parseDouble(req.getParameter("field-b")));
        object.setFieldC(req.getParameter("field-c") != null);

        try {
            object.setId(Integer.parseInt(req.getParameter("id")));
        } catch(NumberFormatException e) {}
        try {

            if(object.getId() == null) {
                Storage.create(object);
            } else {
                Storage.update(object);
            }
        } catch(SQLException e) {
            throw new ServletException(e);
        }
        resp.sendRedirect(req.getContextPath() + "/index.html");
    }
}
```

Изменённый DeleteServlet.java

```
import java.io.IOException;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DeleteServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        for(String id : req.getParameterValues("id")) {
            try {
                Storage.delete(Integer.parseInt(id));
            } catch(NumberFormatException e) {
            } catch(SQLException e) {
                throw new ServletException(e);
            }
        }
        resp.sendRedirect(req.getContextPath() + "/index.html");
    }
}
```

Так как во всех классах-сервлетах исключительная ситуация типа `SQLException` фактически не обрабатывается, а маскируется под `ServletException` и отдаётся на обработку контейнеру сервлетов. Однако страницу с подробной информацией о произошедшем исключении (так называемых stack trace) пользователю отображать нежелательно, необходимо ему в случае такой ошибки отобразить страницу со специальным сообщением. Сделать это можно, создав отдельную JSP-страницу для отображения сообщения об ошибке и зарегистрировав её в дескрипторе развёртывания приложения. Эта часть остаётся на самостоятельное изучение.

4.2. Лабораторная работа № 4 «JDBC»

Модифицировать web-приложение предыдущих частей так, чтобы все данные хранились в базе данных, структура которой должна соответствовать варианту. Реализовать страницу обработки ошибок.

СПИСОК РЕКОМЕНДОВАННЫХ ИСТОЧНИКОВ

1. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес – Санкт-Петербург: Питер, 2014. – 368 с.

2. Фаулер, М. Архитектура корпоративных программных приложений. – Москва: Вильямс, 2007. – 544 с.