ЛАБОРАТОРНАЯ РАБОТА №1.
Создание веб-приложения с использованием Servlet и технологии JSP.

1. Цель работы

Изучить возможности Servlet API и технологии JavaServer Pages для разработки веб-приложений и получить навыки реализации сервлетов и JSP-страниц для обработки запросов пользователей.

2. Методические указания

Лабораторная работа направлена на приобретение навыка написания веб-приложений на языке J2EE.

Требования к результатам выполнения лабораторного практикума:

- изучить теоретический материал и примеры к нему;
- выполнить индивидуальное задание. При выполнении задания необходимо сопровождать текст программы комментариями;
- по завершении выполнения задания составить отчет о проделанной работе.

3. Теоретический материал

Определения:

servlet (сервлет) Java программа, которая расширяет функциональные возможности веб-сервера, динамически генерируя содержание и взаимодействуя с веб-клиентами при помощи принципа запрос-ответ.

servlet container (контейнер сервлета) Контейнер, обеспечивающий сетевые службы, при помощи которых посылаются запросы и ответы, декодируются запросы и форматируются ответы. Все контейнеры сервлетов должны поддерживать HTTP-протокол, но могут также поддерживать дополнительные протоколы, например, HTTPS.

servlet container, distributed (распределенный контейнер сервлета) Контейнер сервлета, запускающий веб-приложения, которые помечены как распределенные и выполняются на нескольких виртуальных машинах Java. При этом виртуальные машины могут быть запущены, как на одном, так и на разных компьютерах.

servlet context (контекст сервлета) Объект, содержащий представление (вид) веб-приложения, в котором запущен сервлет. Используя контекст, сервлет может вести журнал событий, получать URL-ссылки на ресурсы, а также устанавливать и хранить атрибуты, которые могут использоваться другими сервлетами в приложении.

servlet mapping (отображение сервлета) Определяет связь между структурой URL и сервлетом. Используется для отображения запросов в сервлеты. Если контейнер, обрабатывающий запрос, является JSP-контейнером, то неявно отображается URL, содержащий расширение .jsp.

Технология Java Servlet

Технология Java Servlet предоставляет веб-разработчикам простой последовательный механизм для увеличения функциональности веб-сервера и для доступа к существующим коммерческим системам. Сервлеты Java расширяют возможности веб-приложений.

Сервлеты – это компоненты приложений Java Enterprise Edition, выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них. Наибольшее распространение получили сервлеты, обрабатывающие

клиентские запросы по протоколу HTTP. Сервлет может применяться, например, для создания серверного приложения, получающего от клиента запрос, анализирующего его и делающего выборку данных из базы данных, а также пересылающего клиенту страницу HTML, сгенерированную с помощью JSP на основе полученных данных.

Все сервлеты реализуют общий интерфейс Servlet. Для обработки HTTP-запросов можно воспользоваться в качестве базового класса абстрактным классом HttpServlet. Базовая часть классов JSDK помещена в пакет javax.servlet. Однако класс HttpServlet и все, что с ним связано, располагаются на один уровень ниже в пакете javax.servlet.http.

Жизненный цикл сервлета начинается с его загрузки в память контейнером сервлетов при старте либо в ответ на первый запрос. Далее происходят инициализация, обслуживание запросов и завершение существования.

Первым вызывается метод init(). Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов. Чаще всего в этом методе программисты помещают код, кэширующий данные фазы инициализации.

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов. Появившийся запрос обслуживается методом service() сервлета, а все параметры запроса упаковываются в объект ServletRequest, который передается в качестве первого параметра методу service(). Второй параметр метода – объект ServletResponse. В этот объект упаковываются выходные данные в процессе формирования ответа клиенту. Каждый новый запрос приводит к новому вызову метода service(). В соответствии со спецификацией JSDK, метод service() должен уметь обрабатывать сразу несколько запросов, т.е. быть синхронизирован для выполнения в многопоточных средах. Если же нужно избежать множественных запросов, сервлет должен реализовать интерфейс SingleThreadModel, который не содержит ни одного метода и только указывает серверу об однопоточной природе сервлета. При обращении к такому сервлету каждый новый запрос будет ожидать в очереди, пока не завершится обработка предыдущего запроса.

После завершения выполнения сервлета контейнер сервлетов вызывает метод destroy(), в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

Интерфейсом Servlet предусмотрена реализация еще двух методов: getServletConfig() и getServletInfo(). Первый возвращает объект типа ServletConfig, содержащий параметры конфигурации сервлета, а второй – строку, описывающую назначение сервлета.

При разработке сервлетов в качестве базового класса в большинстве случаев используют не интерфейс Servlet, а класс HttpServlet, отвечающий за обработку запросов HTTP.

Класс HttpServlet имеет реализованный метод service(), служащий диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам. В спецификации HTTP определены следующие методы: GET, HEAD, POST, PUT, DELETE, OPTIONS и TRACE. Наиболее часто употребляются методы GET и POST, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.

При использовании метода GET (по умолчанию) параметры передаются как часть URL, значения могут выбираться из полей формы или передаваться непосредственно через URL. При этом запросы кэшируются и имеют ограничения на размер. При использовании метода POST (method=POST) параметры (поля формы) передаются в содержимом HTTP-запроса и упакованы согласно полю заголовка Content-Type. По умолчанию в формате: <имя>=<значение>&<имя>=<значение>&...

Однако форматы упаковки параметров могут быть самые разные, например: в случае передачи файлов с использованием формы enctype="multipart/form-data".

В задачу метода service() класса HttpServlet входит анализ полученного через запрос метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс do: doGet() или doPost(). Кроме

этих методов могут использоваться методы: doHead(), doPut(), doDelete(), doOptions() и doTrace(). Разработчик должен переопределить нужный метод, разместив в нем функциональную логику.

Технология Java Server Pages (JSP)

Сервлеты позволяют получать запросы от клиента, совершать некоторую работу и выводить результаты на экран. Сервлет прекрасно работает до того момента, пока речь идет об обработке информации, т.е. до вывода информации на экран. В сервлет можно вставить достаточно сложную логику, сделать вызовы к базе данных и многое-многое другое, что необходимо для приложения. Но вот осуществлять вывод на экран внутри самого сервлета - очень неудобно.

Технология проектирования **Java Server Pages (JSP)** - это одна из технологий J2EE, которая представляет собой расширение технологии сервлетов для упрощения работы с веб-содержимым. Страницы JSP позволяет легко разделить веб-содержимое на статическую и динамическую часть, допускающую многократное использование ранее определенных компонентов.

Спецификация Java Server Pages наследует и расширяет спецификацию сервлетов. Как и сервлеты, компоненты JSP относятся к компонентам веб и располагаются в веб-контейнере. Страницы JSP не зависят от конкретной реализации веб-контейнера, что обеспечивает возможность их повторного использования.

В дополнение к классам и интерфейсам для программирования сервлетов (пакеты *javax.servlet* и *javax.servlet/http*), в пакетах *javax.servlet.jsp* и *javax.servlet.jsp.target* содержатся классы и интерфейсы, относящиеся к программированию Java Server Pages.

Обзор технологии Java Server Pages

Технология Java Server Pages содержит четыре ключевых компонента:

1. **Директивы** *(directive)* представляют собой сообщения для контейнера JSP, дающим возможность определить параметры страницы, подключения других ресурсов, использовать собственные нестандартные библиотеки тегов.
2. **Действия** *(actions)* инкапсулируют функциональные возможности в предопределенных тегах, которые можно встраивать в JSP-страницу. Действия часто выполняются на основе информации, посылаемой на сервер в составе запроса от определенного клиента. Действия также могут создавать объекты Java для использования их в скриптлетах JSP.
3. **Скриптлеты** *(scriptlets)* позволяют вставлять код Java в страницы JSP, который взаимодействует с объектами страницы при обработке запроса.
4. **Библиотеки тегов** *(tag library)* являются составной частью механизма расширения тегов, допускающего разработку и использование собственных тегов.

Наличие данных с неизменяемой структурой определяют выбор программиста в принятии решения, какую технологию следует использовать: сервлеты или страницы JSP. Программисты предпочитают использовать страницы JSP, если основная часть посылаемого клиенту содержимого представляет собой данные с неизменяемой структурой, и лишь небольшая часть содержимого генерируется динамически с помощью кода Java. Сервлеты предпочтительнее использовать, если только небольшая часть содержимого, посылаемого клиенту, представляет собой данные с неизменяемой структурой. На самом деле отдельные сервлеты могут вообще не генерировать содержимого для клиента, выполняя определенную задачу в интересах клиента, а затем вызывают другие сервлеты или JSP-страницы, чтобы отправить ответ.

Необходимо заметить, что во многих случаях сервлеты и JSP-страницы являются взаимозамеяемыми. Подобно сервлетам, JSP-страницы обычно выполняются на стороне веб-сервера, который называют **контейнером JSP**.

Когда веб-сервер, поддерживающий технологию JSP, принимает первый запрос на JSP-страницу, контейнер JSP транслирует эту JSP-страницу в сервлет Java, который обслуживает текущий запрос и все последующие запросы к этой странице. Если при компиляции нового сервлета возникают ошибки, эти ошибки приводят к *ошибкам на этапе компиляции.* Контейнер JSP на этапе трансляции помещает операторы Java, которые реализует ответ JSP-страницы, в метод *_jspService.* Если сервлет компилируется без ошибок, контейнер JSP вызывает метод *_jspService* для обработки запроса.

JSP-страница может обработать запрос непосредственно, либо вызвать другие компоненты веб-приложения, чтобы содействовать обработке запроса. Любые ошибки, которые возникают в процессе обработки, вызывают *исключительную ситуацию в веб-сервере на этапе запроса*.

Весь статический текст HTML, называемый в документации JSP *шаблоном HTML* (template HTML), сразу направляется в выходной поток. Выходной поток страницы буферизуется. Буферизацию обеспечивает класс *JspWriter*, расширяющий класс *Writer*. Размер буфера по умолчанию ограничен до 8 Кбайт, но его можно изменить атрибутом *buffer* тега *<%@ page>*. Наличие буфера позволяет заносить заголовки ответа в выходной поток совместно с выводимым текстом. В буфере заголовки будут размещены перед текстом.

Таким образом, достаточно написать страницу JSP, сохранить ее в файле с расширением *jsp* и установить файл в контейнер, так же, как и страницу HTML, не заботясь о компиляции. При установке можно задать начальные параметры страницы JSP так же, как и начальные параметры сервлета.

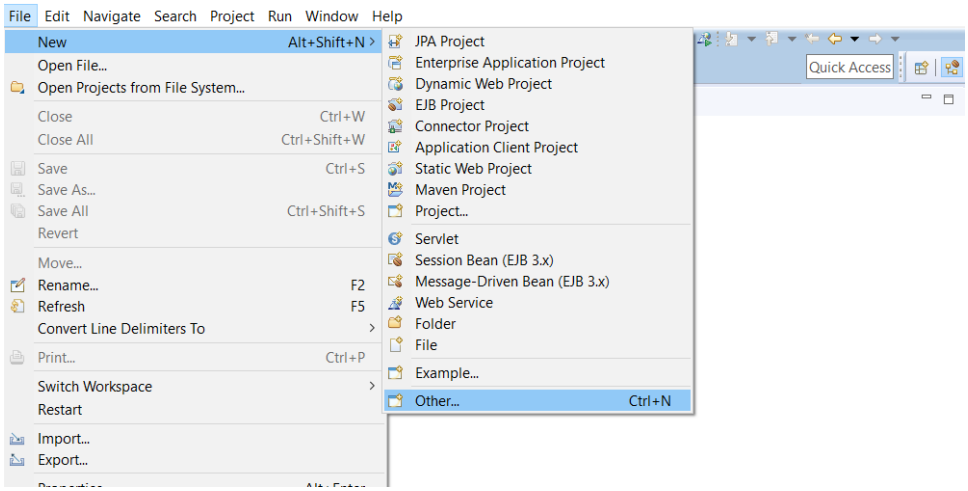
JavaServer Pages (JSP) позволяют отделить динамическую часть страниц от статического HTML. Динамическая часть заключается в специальные теги ***"<% %>":***

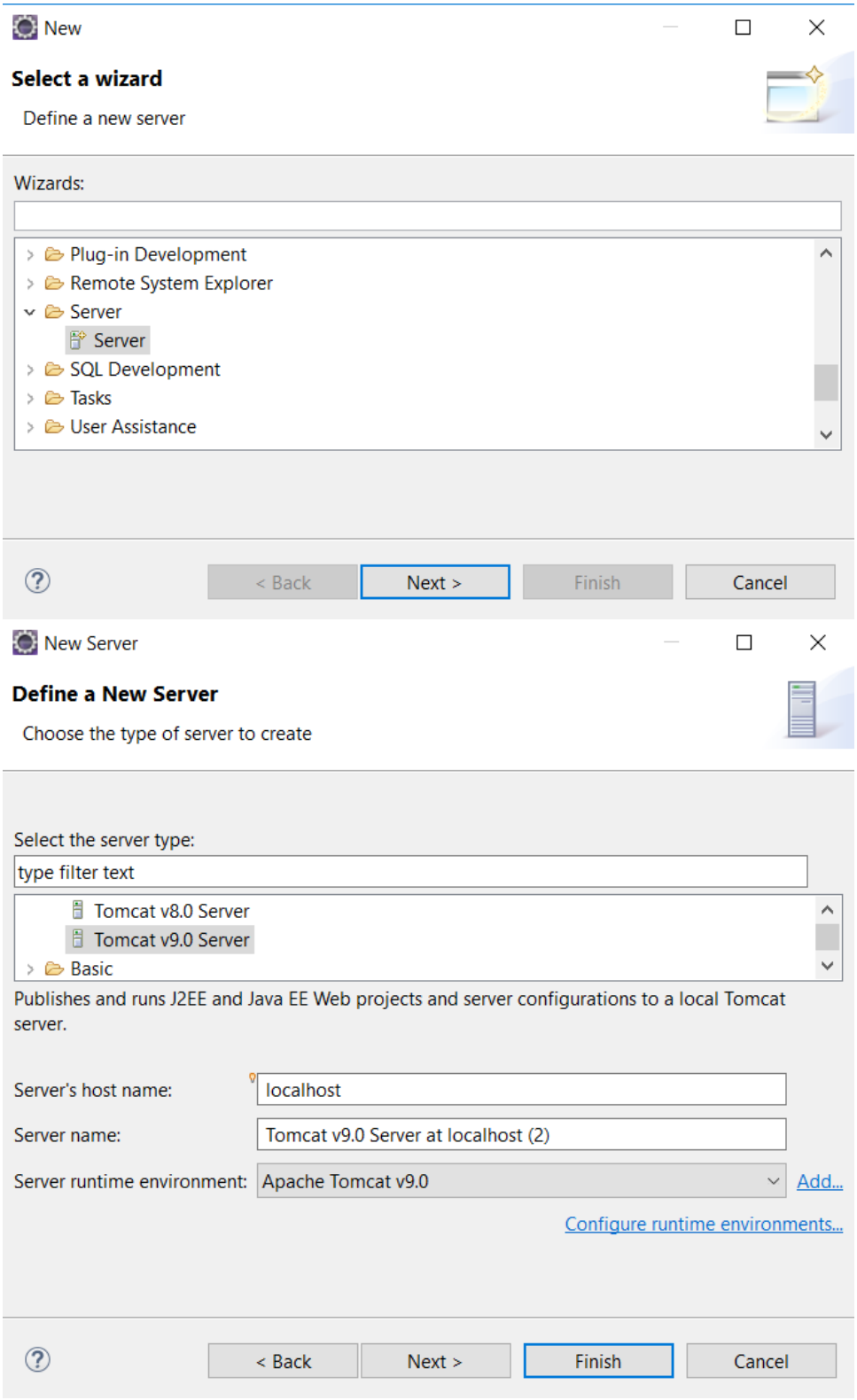
Имя вашего хоста: <%= request.getRemoteHost() %>

Ход работы:

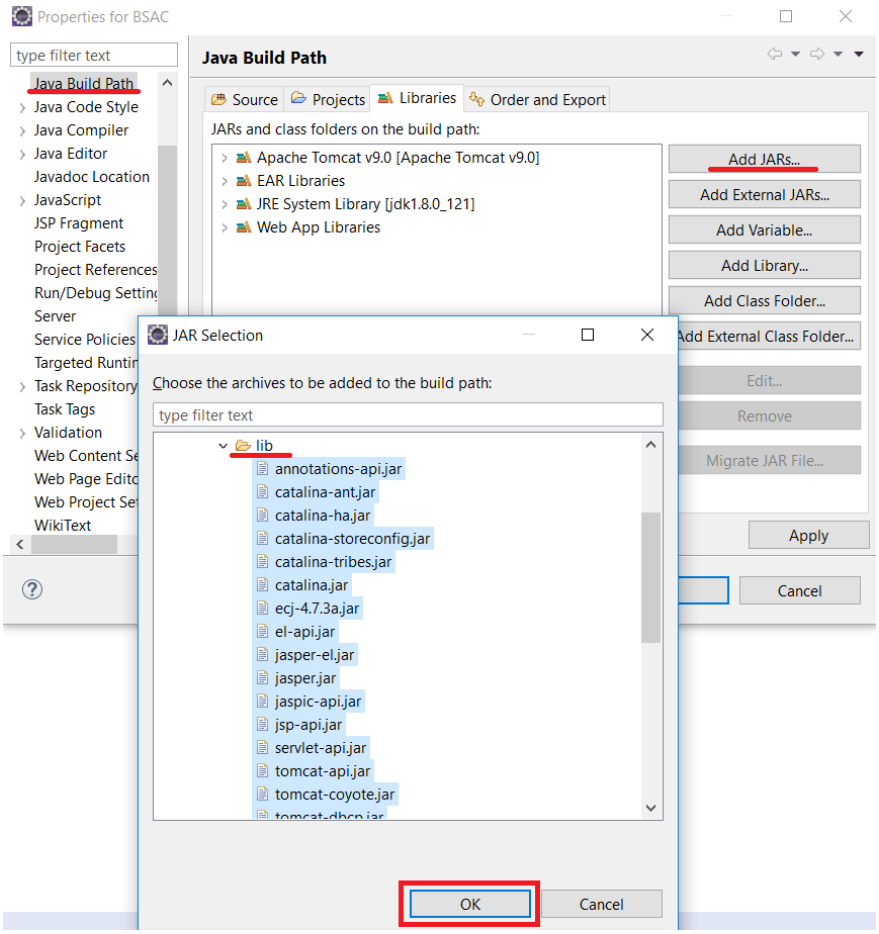
Создание проекта и сервлета.

1. Скачайте и установите **Eclipse IDE for Java EE Developers**.
2. Скачайте и разархивируйте **Apache Tomcat**.
3. Создайте сервер во вкладках **File/New/Other/Server/Server** и подключите Apache Tomcat в качестве сервера.

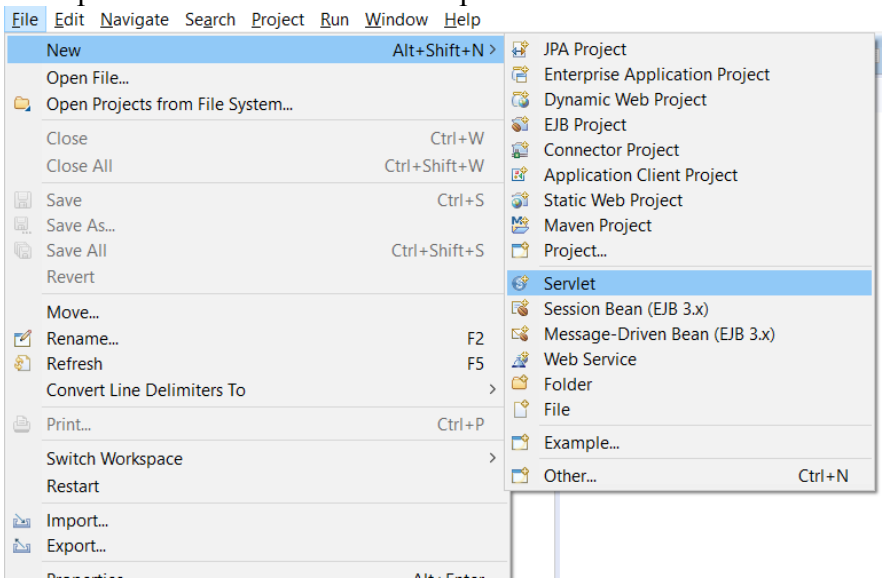


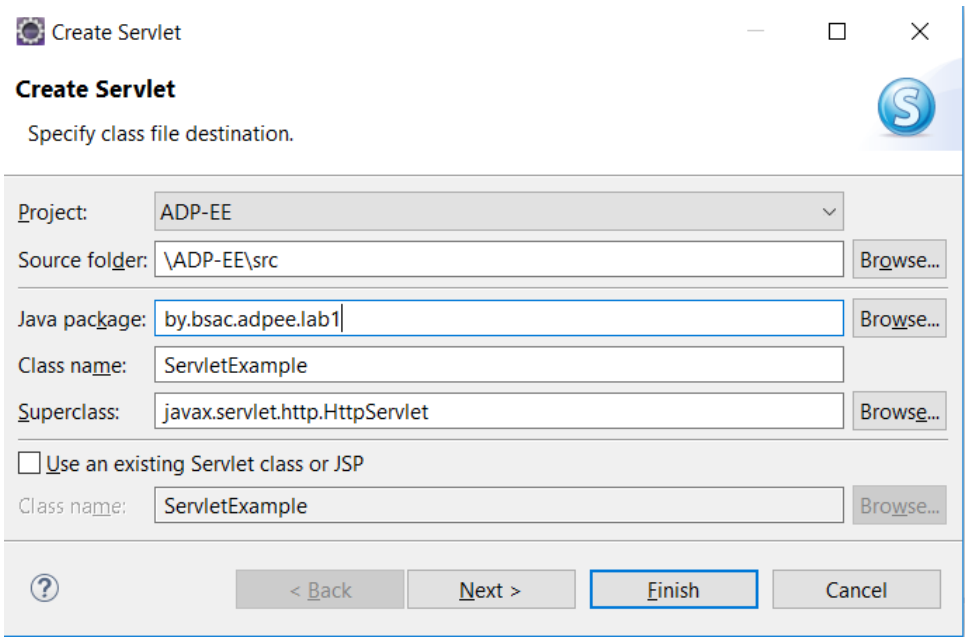


4. Скопируйте все jar файлы с папки lib в Tomcat в папку lib вашего проекта.
5. Зайдите в **Project/Properties/Add JARs** и подключите все библиотеки, которые находятся в lib.

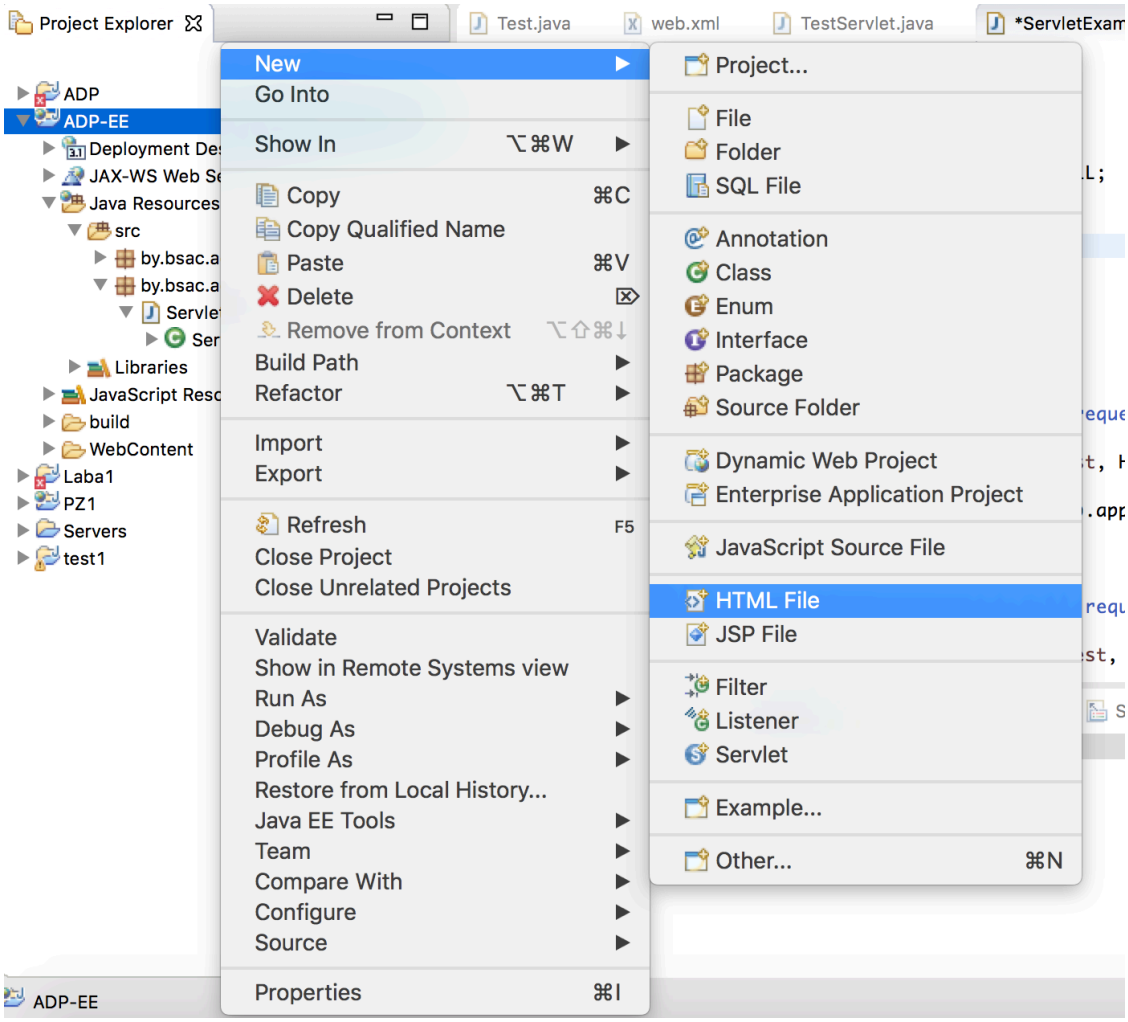


6. Создайте сервлет с именем ServletExample.

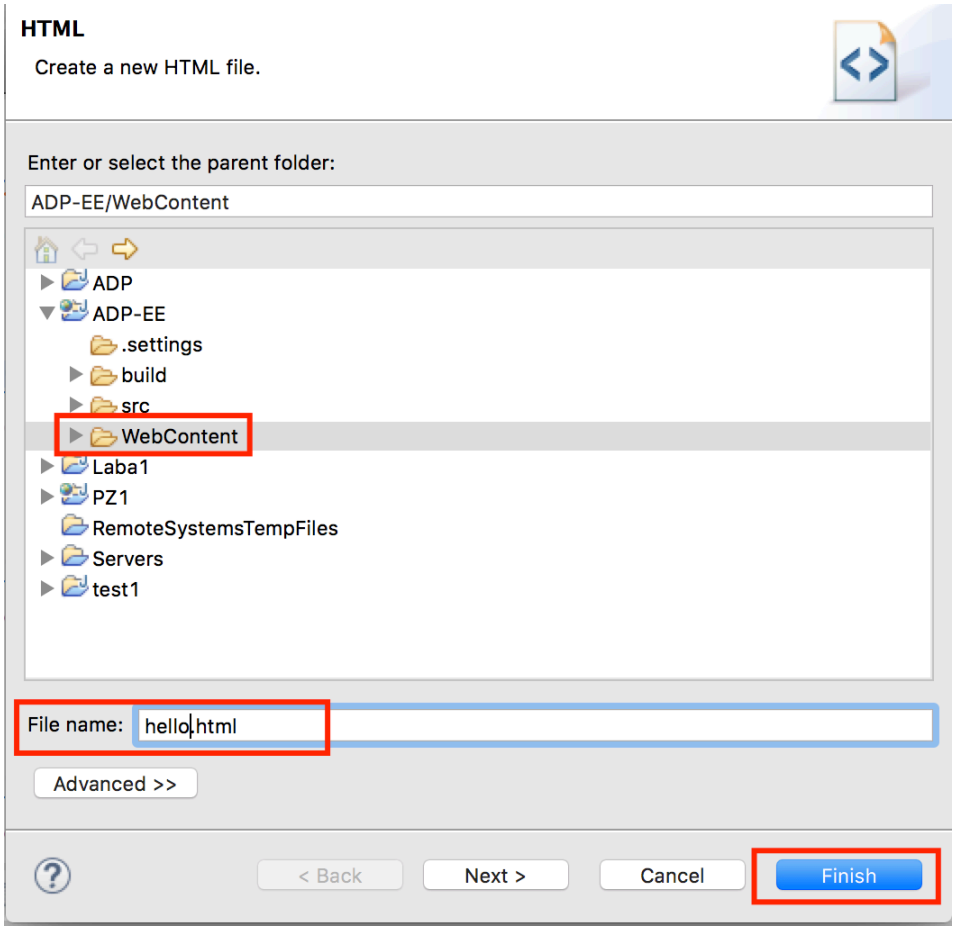




7. Создадим HTML страницу. Для этого выберите пункт меню New-HTML File.



8. Назовите его hello.html.



9. Замените содержимое данного файла на:

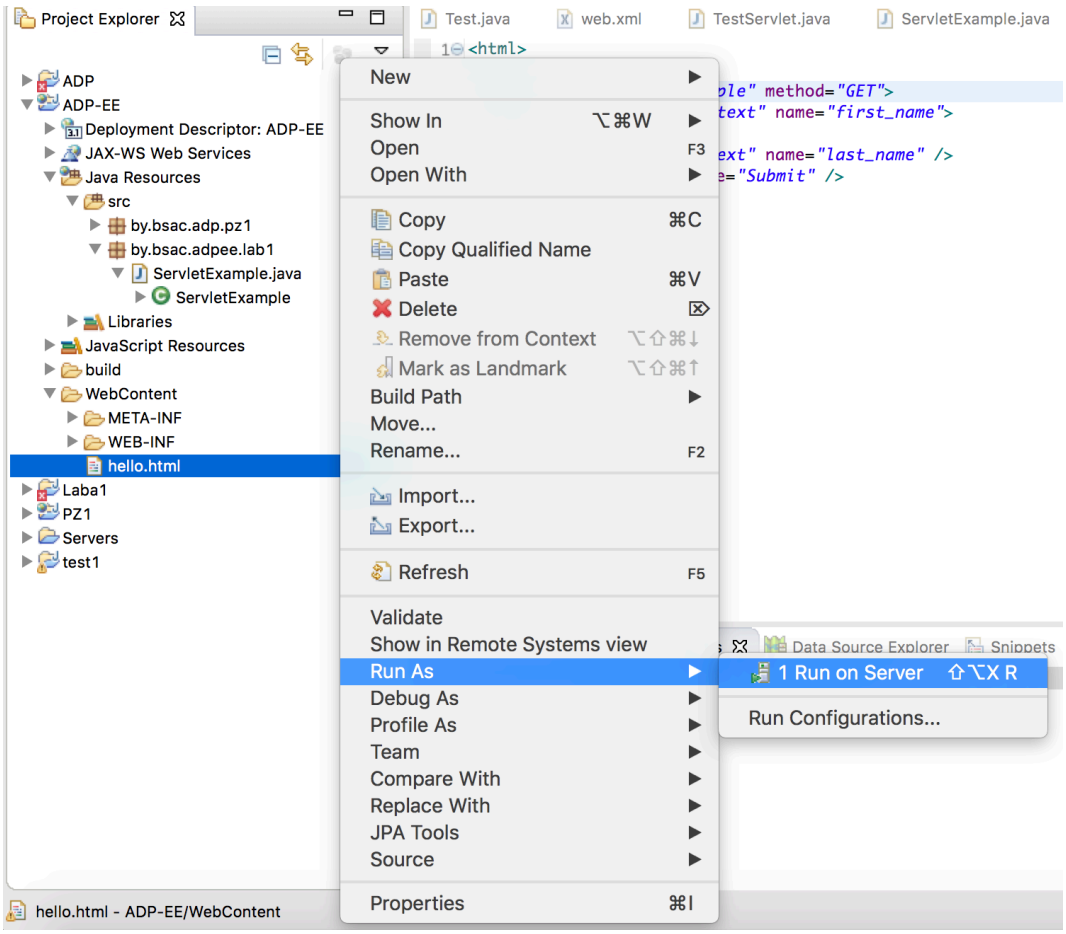
```
<html>
<body>
<h1>Using GET Method to Send Form Data</h1>
<form action="ServletExample" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
<h1>Using POST Method to Send Form Data</h1>
<form action="ServletExample" method="POST">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

10. В программном коде Сервлета внесите изменения в метод doGet и в метод doPost:

```
package by.bsac.adpee.lab1;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ServletExample
 */
@WebServlet("/ServletExample")
public class ServletExample extends HttpServlet {

	private static final long serialVersionUID = 1L;

	/**
	 * @see HttpServlet#HttpServlet()
	 */
	public ServletExample() {
		super();
	}

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
	 *      response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		// Set response content type
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
	String title = "Using " + request.getMethod() + " Method to Read Form Data";

		out.println("</body></html>");
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
	 *      response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		doGet(request, response);
	}

}
```

11. На файле hello.html нажмите правой кнопкой мыши и выберите Run As - Run on Server.



12. Введите имя и фамилию, нажмите кнопку Submit. В результате ваши данные передадутся на сервер с помощью метода GET (обратите внимание на URL) или во втором случае ваши данные передадутся с помощью метода POST (параметры хранятся в теле HTML-запроса).
13. Загрузите браузер и перейдите по адресу http://localhost:8080/ADP-EE/hello.html Выполните действия из пункта 7.

Обработка параметров формы (RadioButton и CheckBox)

1. Допишите содержимое файла hello.html, где введёте дополнительные характеристики Пол - мужской или женский и Работник или Студент.

```
<html>
<body>
<h1>Using GET Method to Send Form Data</h1>
<form action="ServletExample" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<!-- Начало - Обработка checkbox и radiobutton -->
<br />
<input type="radio" name="gender" value="Male" checked/>Male
<br />
<input type="radio" name="gender" value="Female"/>Female
<br/>
<input type="checkbox" name="employee" checked="checked" /> Employee
<br/>
<input type="checkbox" name="student" /> Student
```

```
<br />
<input type="checkbox" name="other" /> Other
<br />
<!-- Конец - Обработка checkbox и radiobutton -->
<input type="submit" value="Submit" />
</form>
<h1>Using POST Method to Send Form Data</h1>
<form action="ServletExample" method="POST">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<!-- Начало - Обработка checkbox и radiobutton -->
<br />
<input type="radio" name="gender" value="Male" checked/>Male
<br />
<input type="radio" name="gender" value="Female"/>Female
<br />
<input type="checkbox" name="employee" checked="checked" /> Employee
<br />
<input type="checkbox" name="student" /> Student
<br />
<input type="checkbox" name="other" /> Other
<br />
<!-- Конец - Обработка checkbox и radiobutton -->
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

2. Внесите дополнения в программный код Сервлета.

```
package by.bsac.adpee.lab1;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ServletExample
 */
@WebServlet("/ServletExample")
public class ServletExample extends HttpServlet {
	public static String GENDER_MALE = "male";
	public static String GENDER_FEMALE = "female";

	private static final long serialVersionUID = 1L;

	/**
	 * @see HttpServlet#HttpServlet()
	 */
```

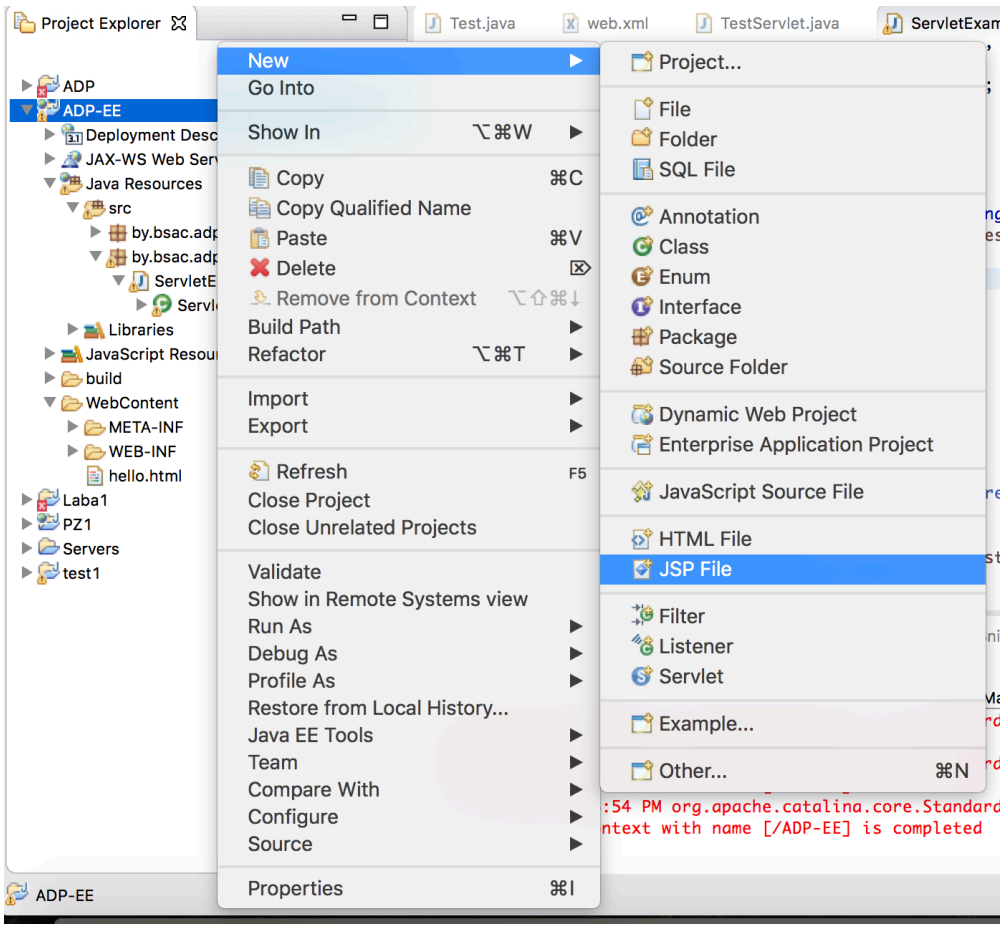
```
	public ServletExample() {
		super();
	}

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
	 *      response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		// Set response content type
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
	String title = "Using " + request.getMethod() + " Method to Read Form Data";

		// опеределение пола - обработка radio button
		String genderHtml =
(request.getParameter("gender").equalsIgnoreCase(GENDER_MALE))
				? ("<li><b>Gender:</b>" + GENDER_MALE + "</li>") :
("<li><b>Gender:</b> " + GENDER_FEMALE + "</li>");

		String docType = "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";
		out.println(docType + "<html>\n" + "<head><title>" + title + "</title></head>\n"
				+ "<body bgcolor=\"#f0f0f0\">\n" + "<h1 align=\"center\">" + title
+ "</h1>\n" + "<ul>\n"
				+ "  <li><b>First Name</b>: " +
request.getParameter("first_name") + "\n" + "  <li><b>Last Name</b>: "
				+ request.getParameter("last_name") + "\n" +
				// определение пола
				genderHtml + "\n" +
				// определение пола

				// обработка checkbox
				"  <li><b>Student : </b> " + request.getParameter("student") + "\n"
+ "  <li><b>Employee: </b> "
				+ request.getParameter("employee") + "\n" + "  <li><b>Other:
</b> " + request.getParameter("other")
				+ "\n</ul>\n");

		// вывести все параметры - начало
		out.println("<table width=\"100%\" border=\"1\" align=\"center\">\n" + "<tr
bgcolor=\"#949494\">\n"
				+ "<th>Param Name</th><th>Param Value(s)</th>\n" +
"</tr>\n");

		Enumeration paramNames = request.getParameterNames();

		while (paramNames.hasMoreElements()) {
			String paramName = (String) paramNames.nextElement();
			out.print("<tr><td>" + paramName + "</td>\n<td>");
			String[] paramValues = request.getParameterValues(paramName);
```

```
                        // Read single valued data
                        if (paramValues.length == 1) {
                                String paramValue = paramValues[0];
                                if (paramValue.length() == 0)
                                        out.println("<i>No Value</i>");
                                else
                                        out.println(paramValue);
                        } else {
                                // Read multiple valued data
                                out.println("<ul>");
                                for (int i = 0; i < paramValues.length; i++) {
                                        out.println("<li>" + paramValues[i]);
                                }
                                out.println("</ul>");
                        }
                }
                out.println("</tr>\n</table>\n");
                // вывести все параметры - конец

                out.println("</body></html>");
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
         *      response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
                doGet(request, response);
        }

}
```

3. Загрузите браузер и перейдите по адресу http://localhost:8080/ADP-EE/hello.html

JSP

1. Создадим JSP - страницу welcome.jsp



2. Допишите в содержимое hello.html текст программы:

```
<html>
<body>
<h1>Using GET Method to Send Form Data</h1>
<form action="ServletExample" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<!-- Начало - Обработка checkbox и radiobutton -->
<br />
<input type="radio" name="gender" value="Male" checked/>Male
<br />
<input type="radio" name="gender" value="Female"/>Female
<br/>
<input type="checkbox" name="employee" checked="checked" /> Employee
<br/>
<input type="checkbox" name="student" /> Student
<br />
<input type="checkbox" name="other" /> Other
<br />
<!-- Конец - Обработка checkbox и radiobutton -->
<input type="submit" value="Submit" />
</form>
<h1>Using POST Method to Send Form Data</h1>
<form action="ServletExample" method="POST">
First Name: <input type="text" name="first_name">
```

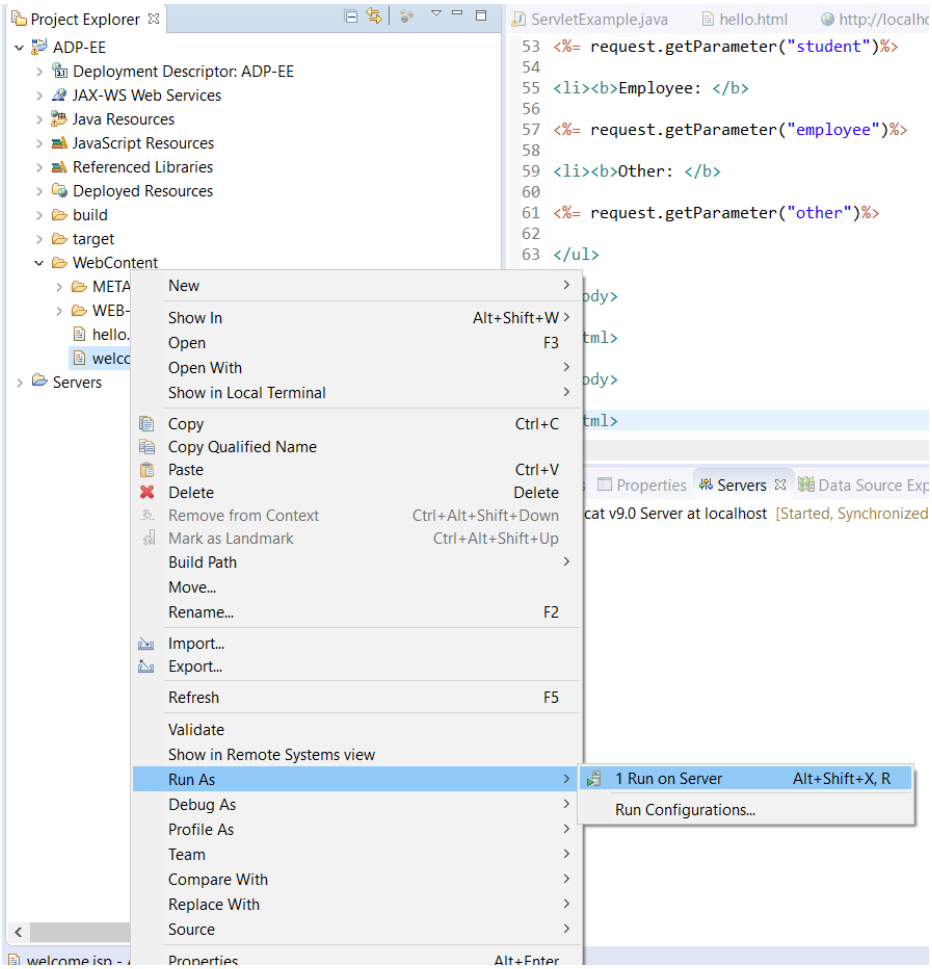
```
<br />
Last Name: <input type="text" name="last_name" />
<!-- Начало - Обработка checkbox и radiobutton -->
<br />
<input type="radio" name="gender" value="Male" checked/>Male
<br />
<input type="radio" name="gender" value="Female"/>Female
<br />
<input type="checkbox" name="employee" checked="checked" /> Employee
<br />
<input type="checkbox" name="student" /> Student
<br />
<input type="checkbox" name="other" /> Other
<br />
<!-- Конец - Обработка checkbox и radiobutton -->
<input type="submit" value="Submit" />
</form>

<h1>Welcome JSP</h1>
<form action="welcome.jsp" method="POST">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<!-- Начало - Обработка checkbox и radiobutton -->
<br />
<input type="radio" name="gender" value="Male" checked/>Male
<br />
<input type="radio" name="gender" value="Female"/>Female
<br />
<input type="checkbox" name="employee" checked="checked" /> Employee
<br />
<input type="checkbox" name="student" /> Student
<br />
<input type="checkbox" name="other" /> Other
<br />
<!-- Конец - Обработка checkbox и radiobutton -->
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

3. В welcome.jsp разместите текст программы:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<html>
<body>
<html>
<head>
<title>Using JSP/POST Method to Read Form Data</title>
</head>
<body>
<h1>Using JSP/POST Method to Read Form Data</h1>
```

```
<ul>
<li><b>First Name:</b>
  <%= request.getParameter("first_name")%>
</li>
<li><b>Last  Name:</b>
  <%= request.getParameter("last_name")%>
</li>
<%
String genderParam = request.getParameter("gender");
if (genderParam != null) {
String genderHtml = (genderParam.equalsIgnoreCase("male"))
                                    ? ("<li><b>Gender:</b> Male</li>") : ("<li><b>Gender:</b>
Female</li>");
out.println(genderHtml);
}
%>
<li><b>Student: </b>
<%= request.getParameter("student")%>
<li><b>Employee: </b>
<%= request.getParameter("employee")%>
<li><b>Other: </b>
<%= request.getParameter("other")%>
</ul>
</body>
</html>
</body>
</html>
```

4. На файле welcome.jsp нажмите правой кнопкой мыши и выберите Run As - Run on Server.



5. Загрузите браузер и перейдите по адресу http://localhost:8080/ADP-EE/hello.html

4. Порядок выполнения работы

- изучить теоретический материал;
- выполнить задания из теоретической части лабораторной работы;
- разработайте веб-приложение, использующее сервлет для поиска информации о сотрудниках организации. Данные о сотрудниках хранятся в таблице Employee. Для осуществления поиска пользователь указывает фамилию сотрудника и просматривает информацию о найденных сотрудниках (возможно существование нескольких сотрудников с одинаковыми фамилиями).

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект
2. Создать таблицу employee и заполнить ее данными
3. Разработать сервлет, который выбирает из БД записи, соответствующие запросу пользователя и отображает результат.
4. Упаковать приложение и развернуть на сервере.
5. Протестировать работу приложения в браузере

Необходимо расширить функциональные возможности приложения поиска и просмотра сотрудников возможностью добавления новых сотрудников. Прежде всего, необходимо изменить код главной страницы таким образом, чтобы при отсутствии строки поиска в таблице отображались все сотрудники. Далее, необходимо создать новую страницу, содержащую форму для ввода информации о новом сотруднике и создать на нее ссылку на главной странице. Разработать сервлет для обработки параметров нового сотрудника и

создания записи в БД. После создания сотрудника главная страница автоматически обновляется.
- выполнить индивидуальные задания в соответствии с вашим вариантом.

5. Индивидуальные задания

Создать сервлет и взаимодействующие с ним пакеты Java-классов и HTML-документов. Готовое веб-приложение разместить на сервере Tomcat.

1. Генерация таблиц по переданным параметрам: заголовок, количество строк и столбцов, цвет фона.
2. Вычисление тригонометрических функций в градусах и радианах с указанной точностью. Выбор функций должен осуществляться через выпадающий список.
3. Поиск слова, введенного пользователем. Поиск и определение частоты встречаемости осуществляется в текстовом файле, расположенном на сервере.
4. Вычисление объемов тел (параллелепипед, куб, сфера, тетраэдр, тор, шар, эллипсоид и т.д.) с точностью и параметрами, указываемыми пользователем.
5. Поиск и (или) замена информации в коллекции по ключу (значению).
6. Выбор текстового файла из архива файлов по разделам (поэзия, проза, фантастика и т.д.) и его отображение.
7. Выбор изображения по тематике (природа, автомобили, дети и т.д.) и его отображение.
8. Информация о среднесуточной температуре воздуха за месяц задана в виде списка, хранящегося в файле. Определить: а) среднемесячную температуру воздуха; б) количество дней, когда температура была выше среднемесячной; в) количество дней, когда температура опускалась ниже ; г) три самых теплых дня.
9. Реализация адаптивного теста из цепочки в 3 – 4 вопроса.
10. Вывод фрагментов текстов шрифтами различного размера. Размер шрифта и количество строк задается на стороне клиента.
11. Информация о точках на плоскости хранится в файле. Выбрать все точки, наиболее приближенные к заданной прямой. Параметры прямой и максимальное расстояние от точки до прямой вводятся на стороне клиента.
12. Осуществить сортировку введенного пользователем массива целых чисел. Числа вводятся через запятую.
13. Осуществить форматирование выбранного пользователем текстового файла, так чтобы все абзацы имели отступ ровно 3 пробела, а длина каждой строки была ровно 80 символов и не имела начальными и конечными символами пробел.
14. В тексте нет слов, начинающихся одинаковыми буквами. Напечатать слова текста в таком порядке, чтобы последняя буква каждого слова совпадала с первой буквой последующего слова. Если все слова нельзя напечатать в таком порядке, найти такую цепочку, состоящую из наибольшего количества слов.
15. Найти наибольшее количество предложений текста, в которых есть одинаковые слова.
16. Найти такое слово в первом предложении, которого нет ни в одном из остальных предложений.
17. Во всех вопросительных предложениях текста найти и напечатать без повторений слова заданной длины.
18. В каждом предложении текста поменять местами первое слово с последним, не изменяя длины предложения.

19. В предложении из n слов первое слово поставить на место второго, второе – на место третьего, и т.д., (n-1)-е слово – на место n-го, n-е слово поставить на место первого. В исходном и преобразованном предложениях между словами должны быть или один пробел, или знак препинания и один пробел.
20. Текст шифруется по следующему правилу: из исходного текста выбирается 1, 4, 7, 10-й и т.д. (до конца текста) символы, затем 2, 5, 8, 11-й и т.д. (до конца текста) символы, затем 3, 6, 9, 12-й и т.д. Зашифровать заданный текст.
21. На основании правила кодирования, описанного в предыдущем примере, расшифровать заданный набор символов.
22. Напечатать слова русского текста в алфавитном порядке по первой букве. Слова, начинающиеся с новой буквы, печатать с красной строки.
23. Рассортировать слова русского текста по возрастанию доли гласных букв (отношение количества гласных к общему количеству букв в слове).
24. Слова английского текста, начинающиеся с гласных букв, рассортировать в алфавитном порядке по 1-й согласной букве слова.
25. Все слова английского текста рассортировать по возрастанию количества заданной буквы в слове. Слова с одинаковым количеством расположить в алфавитном порядке.
26. Ввести текст и список слов. Для каждого слова из заданного списка найти, сколько раз оно встречается в тексте, и рассортировать слова по убыванию количества вхождений.
27. Все слова текста рассортировать в порядке убывания их длин. При этом все слова одинаковой длины рассортировать в порядке возрастания в них количества гласных букв.
28. Удалить из текста его часть, заключенную между двумя символами, которые вводятся (например между скобками ‘(’ и ‘)’ или между звездочками ‘*’ и т.п.).
29. Найти, каких букв, гласных или согласных, больше в каждом предложении текста.
30. В стихотворении найти количество слов, начинающихся и заканчивающихся гласной буквой.

7. Контрольные вопросы

1. Что такое сервлет?
2. Что такое контейнер сервлета? Жизненный цикл сервлета.
3. Назовите преимущества и недостатки методов post и get.
4. Для чего применяется технология JSP?