LS/EECS
Building E-commerce Applications
LAB 1

Objectives

- understand the servlet session
- develop a simple client-server application

The Persistence Layers

(for reading)

Servlets come with three scopes that allow you to store data at various levels and for various durations:

- The Context (Application) Scope stores data for all clients as long as the server is running. You access it using the get/set attribute methods of the ServletContext object (accessible from the servlet).

```
// Get the ServletContext object
ServletContext context = getServletContext();

// Store a value in the ServletContext using setAttribute method
context.setAttribute("message", "Hello from StoreServlet");

// Retrieve the value from the ServletContext using getAttribute method
String message = (String) context.getAttribute("message");
```

- **The Session Scope** stores data per client as long as the session of the client has not expired. You access it using the get/set attribute methods of the `HttpSession` object (accessible in `doGet`/`Post` from request). Read the API of request object to see what else can be done with it.

```
// Get the HttpSession object
HttpSession session = request.getSession();

// Store a value in the HttpSession using setAttribute method
session.setAttribute("name", "John Smith");

// Retrieve the value from the HttpSession using getAttribute method
String name = (String) session.getAttribute("name");
```

- **The Request Scope** stores data per client as long as the response of the current request has not been sent yet. You access it using the get/set attribute methods of the

`ServletRequest` object (accessible in `doGet`/`Post` from request).

```
// Get the ServletRequest object
ServletRequest req = request;

// Store an attribute in the ServletRequest using setAttribute method
req.setAttribute("message", "Hello from StoreRequestServlet");

// Retrieve the attribute from the ServletRequest using getAttribute
method
String message = (String) req.getAttribute("message");
```

NOTES:

- All three scopes use `setAttribute(name,value)` and `getAttribute(name)` methods to set and access the attributes
- All three scopes use a key-value map to store these attributes, and hence, they are typed.
- Always ask yourself where does it make sense to store an attribute, in request, session or context?
- Attributes are different than "Parameters." Parameters are part of the query string that comes from clients, they are strings and can be accessed with "`getParameter(name)`" method of HTTP Request object.

Task A: A simple backend service: Shopping cart Price Calculator

The objective of this step is to create the main application that can calculate the total price that a customer should pay for the goods they bought. You will create a new dynamic web project in Eclipse, a new HTML file that contains a form to get the input values from the user, and a new servlet that handles the GET request from the form and calculates and displays the total price using Java code.

1. Open Eclipse and create a new dynamic web project named `LastName_v2`. Choose Tomcat as the target runtime and select the default configuration.
    - To create a new dynamic web project, go to File -> New -> Dynamic Web Project.
    - To choose Tomcat as the target runtime, click on New Runtime... and select Apache Tomcat from the list. Then browse to the installation directory of Tomcat and click on Finish.
    - To select the default configuration, click on Next until you reach the Web Module page. Then click on Finish.

2. In the `src` folder of your project, create a new servlet named `CalculatorServlet`. This servlet will handle the GET requests and calculate the total price. Write some Java code to implement the `doGet` method of the servlet.

    The method should do the following:

- Get the query parameters from the request object using the `getParameter` method. The parameters are `noItems`, `price`, and `tax`. You may need to parse them into numeric values using methods such as `Integer.parseInt` or `Double.parseDouble`. To get the query parameters from the request object, use the `getParameter` method with the parameter name as an argument:

```
String noItemsParam = request.getParameter("noItems");
```

  To parse them into numeric values, use methods such as `Integer.parseInt` or `Double.parseDouble` with the parameter value as an argument:

```
int noItems = Integer.parseInt(noItemsParam);
```

- Calculate the total price using the formula: $total = noItems * price * \left(1 + tax-100\right)$.

```
double total = noItems * price * (1 + tax / 100);
```

  You may need to round the result to two decimal places using methods such as `Math.round` or `String.format`:

```
total = Math.round(total * 100) / 100.0;
```

- Set the content type of the response object to text/html using the `setContentType` method.

```
response.setContentType("text/html");
```

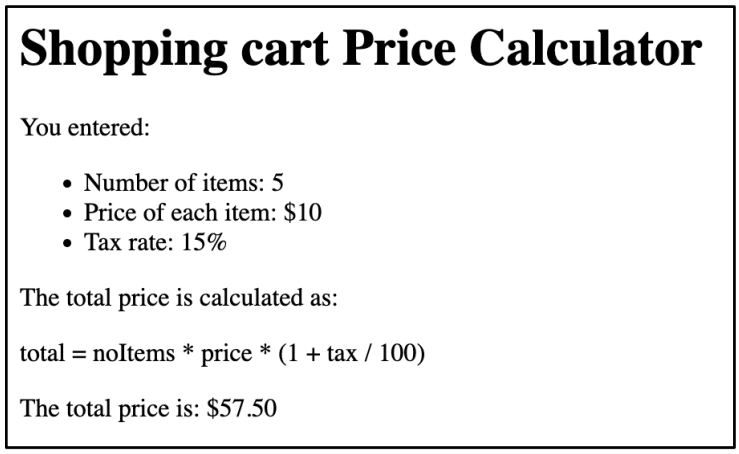
- Get a print writer from the response object using the `getWriter` method:

```
PrintWriter out = response.getWriter();
```

- Write some HTML code to display the total price to the output using the print writer. You may also include some information about the input values and the calculation formula:

```
out.println("<html>");
out.println("<head>");
out.println("<title>Shopping cart Price Calculator</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Shopping cart Price Calculator</h1>");
out.println("<p>You entered:</p>");
out.println("<ul>");
out.println("<li>Number of items: " + noItems + "</li>");
out.println("<li>Price of each item: $" + price + "</li>");
out.println("<li>Tax rate: " + tax + "%</li>");
out.println("</ul>");
out.println("<p>The total price is calculated as:</p>");
out.println("<p>total = noItems * price * (1 + tax / 100)</p>");
out.println("<p>The total price is: $" + total + "</p>");
out.println("</body>");
out.println("</html>");
```

Here is an example output:



3. In the `WEB-INF` folder of your project, open the `web.xml` file. This file contains the configuration of your web application. Write some XML code to map the servlet class `CalculatorServlet` to the URL pattern `/CalculatorServlet`. This will enable the web server to invoke the servlet when the form is submitted.

   To map a servlet class to a URL pattern, use `<servlet>` and `<servlet-mapping>` tags with subtags such as `<servlet-name>`, `<servlet-class>`, and `<url-pattern>`.

```
<web-app>
      <servlet>
            <servlet-name>CalculatorServlet</servlet-name>
            <servlet-class>CalculatorServlet</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>CalculatorServlet</servlet-name>
            <url-pattern>/CalculatorServlet</url-pattern>
      </servlet-mapping>
 </web-app>
```

Task B: Default Tax Rate

The objective of this step is to add a default value for the tax rate parameter in case it is missing from the request. You will define a context parameter named `defaultTaxRate` in the `web.xml` file and assign it a value of `13`. You will also modify the servlet code to get the default tax rate value from the `ServletContext` object using the `getInitParameter` method and use it in the calculation if the tax rate parameter is null or empty.

In the `WEB-INF` folder of your project, open the `web.xml` file. This file contains the configuration of your web application. Write some XML code to define a context parameter named `defaultTaxRate` and assign it a value of `13`. This will set a default value for the tax

rate parameter in case it is missing from the request.

To define a context parameter, use a `<context-param>` tag with subtags such as `<param-name>` and `<param-value>`:

```
<web-app>
      <!-- ... -->
      <context-param>
            <param-name>defaultTaxRate</param-name>
            <param-value>13</param-value>
      </context-param>
 </web-app>
```

Modify the `CalculatorServlet` class in the `src` folder of your project to:
get the default tax rate value from the `ServletContext` object using the `getInitParameter` method with a key of `defaultTaxRate` if the tax rate parameter is missing. Parse this value into a `double` value and store it in the local variable named `tax` and use it for the calculations.

- To get the default tax rate value from the `ServletContext` object, use the `getInitParameter` method with "`defaultTaxRate`" as an argument. This method returns the value of the context parameter that matches the given name, or `null` if there is no such parameter:

  ```
  String defaultTax =
  getServletContext().getInitParameter("defaultTaxRate");
  ```

- To parse this value into a double value and store it in a local variable named `tax`, use methods such as `Double.parseDouble` with `defaultTax` as an argument:

  ```
  double tax = Double.parseDouble(defaultTax);
  ```

Task C: Remember Tax Rate

The objective of this step is to use session state to store and retrieve the tax rate parameter across multiple requests. You will use the `HttpSession` object to store the tax rate parameter as an attribute in the session scope using the `setAttribute` method. You will also modify the servlet code to check if there is an attribute named `taxRate` in the `HttpSession` object using the `getAttribute` method and use its value in the calculation if the tax rate parameter is `null` or empty.

Modify the `CalculatorServlet` class in the `src` folder of your project to:

1. If the tax rate parameter is not `null` or empty, parse it into a double value and store it in the local variable named tax. Also, store this value as an attribute in the `HttpSession` object using the `setAttribute` method with a key of `taxRate`.

- To get the `HttpSession` object from the request object, use the `getSession` method without any arguments.
  ```
  HttpSession session = request.getSession();
  ```

- To store the tax rate value as an attribute in `HttpSession` object with a key of "`taxRate`", use methods such as `setAttribute` with "`taxRate`" and tax as arguments:
  ```
  session.setAttribute("taxRate", tax);
  ```
- To get the tax rate value after storing it as an attribute in the `HttpSession` object"

  ```
  (double) session.getAttribute("taxRate");
  ```

- If there is no attribute named `taxRate` in the `HttpSession` object, get the default tax rate value from the `ServletContext` object using the `getInitParameter` method with a key of `defaultTaxRate`. Parse this value into a `double` value and store it in the local variable named `tax` and use it for the calculations.

Run Curl Commands to test your web application

Here are some examples of how to use CURL to test the application:

- A GET request with valid query parameters:
  ```
  curl "http://localhost:8080/Smith-v2/CalculatorServlet?noItems=5&price=10&tax=15"
  ```
  This command should return an HTML page that displays the total price as $57.50.

- A GET request with missing query parameters:
  ```
  curl "http://localhost:8080/Smith-v2/CalculatorServlet?noItems=5&price=10"
  ```
  This command should return an HTML page that displays the total price as $56.50, using the default tax rate of 13% from the context parameter.

- A GET request with invalid query parameters:
  ```
  curl "http://localhost:8080/Smith-v2/CalculatorServlet?noItems=abc&price=10&tax=15"
  ```
  This command should return an error page that shows an exception message such as `java.lang.NumberFormatException`: For input string: "abc".

- A sequence of commands to test the `HttpSession` feature by sending multiple requests with different or missing tax rate parameters and observing how the servlet uses the stored value from the session attribute or the default value from the context parameter:

  ```
  # Send a GET request with a tax rate of 15%
  curl "http://localhost:8080/Smith-v2/CalculatorServlet?noItems=5&price=10&tax=15"
  ```

```
# Send another GET request with a missing tax rate
curl "http://localhost:8080/Smith-
v2/CalculatorServlet?noItems=5&price=10"

# Send another GET request with a different tax rate of 20%
curl "http://localhost:8080/Smith-
v2/CalculatorServlet?noItems=5&price=10&tax=20"
```

- To be able to use sessions with curl, you **need to use –c and –b options with curl.** Use –c for the first curl command to write all cookies after a completed operation. Curl writes all cookies from its in-memory cookie storage to the given file at the end of operations. If no cookies are known, no data is written. Then use –b for the next sequences of curl commands to use the session created. Example:

```
#first create session.txt file in the directory where you are running
your command.

#For MacOS/Unix-based:

touch session.txt

#For Windows command line

type nul > session.txt

# Send GET request with sessions

curl –c session.txt "http://localhost:8080/Smith-
v2/CalculatorServlet?noItems=5&price=10&tax=15"

curl –b session.txt "http://localhost:8080/Smith-
v2/CalculatorServlet?noItems=5&price=10&"
```

  The first command should return an HTML page that displays the total price as $57.50 and store the tax rate of 15% in the session attribute. The second command should return an HTML page that displays the total price as $57.50, using the stored tax rate of 15% from the session attribute. The third command should return an HTML page that displays the total price as $60.00 and update the tax rate of 20% in the session attribute.