Assignment -3

1. To implement a feature in web application using Java Servlets that track the number of accesses by a client within a single session.

Servlet code:

```
import Java.io. IOException;
import Java.io. printwriter;
import Java.util. Date;
import Javax. servlet. ServletException;
import Javax. Servlet. annotation. webServlet;
import Javax. Servlet. http. HttpServlet;
import Javax. Servlet. http. Http ServletRequest;
import Javax. http. HTTP Servlet Response;
import Javax. Servlet. http. HTTPSession;

public class SessionTrackerServlet extends Http Servlet{
    private static final long serialVersionID=1L;

protected void doGet (HttpServlet Request, Response) {
    printwriter out= response. getwriter();

  HTTP Session = request. get Session (true);

   Integer access Count = (Integer) Session. get Attribute

      if(access count==null){
        access count = 0;
access count ++;

Session . Set Attribute ("access Count", access count);

String sessionEnd = session. getId();
long CreationTime = Session. get createdTime();
long lastAccessTime = session. get last AccessedTime();

out. println ("<html><body>");
out. println ("<h1> Session Tracking Example </h1>");
out. println ("<p> Acess Count: " + access Count);
out. println ("<p> Session creation-Time: " + new Date (creationTime));
out. println("<P> last AccessedTime: " + new Date (lastTime));
out. println ("</body></html> ");
}
}
```

2. Write a Scenario where you had to use JSTL to solve a complex problem and how you went about it. Also, elaborate the function library in JSTL.

Here, the example of how I used JSTL to solve the problem.

```
<%@ taglib prefix="fmt" uri="(Data)" %>
<%@ taglib prefix="fn" uri="(Data)" %>
<c:Set var="price" value="100.00"/>
<c: Set var="discount" value="10.00"/>
<c: set var="exchangedRate" value="1.20"/>
<c: Set var="finalprice" value="${price - (price * discount/100}"/>
<c! set var="finalpriceInUSD" value="${:fn convertCurrency(final
    price, exchangedRate)}"/>
<fmt : formatNumber value="${finalpriceINUSD}" type="currency".
    currency Code="USD"/>.
```

Here, the example of how to create a custom function.

```
import Javax.servlet.jsp.tagext.FunctionInfo;
import Javax.servlet.jsp.tagext.TagLibraryInfo;
public class customfunction{
      public static double convert currency (double price,
               double exchangeRate)
    {
         return price* exchangeRate;
      }
  }
```

To use this custom page in JSP fumctis.

```
<%@ taglib prefix= "fn" uri= "(Data)" %>
<c: set var= "finalPriceUSD" value ="${fn: convert currency
    (finalPrice, exchangeRate)}"/>.
```

3. A page of stock market quotes uses Script to refresh the page every five minutes in order to ensure the lastest stastics remain available.

Here is an example of how to acheive this using Java Script and HTML

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Stock Market Quotes </title>
    <Script src ="script.js"></script>
</head>
<body>
    <!-- Your stock market quotes content here -->
</body>
</html>
```

JavaScript:

```
let refreshInterval= 5*60 * 1000;
let warningTime =20* 1000;
let timer= setInterval (refreshPage, refreshInterval);
let warning= setInterval (showwarning, refreshInterval - warningTime
function refreshPage() {
    window.location.reload();
}
function showWarning(){
    if( confirm ("The page will refresh in 20sec")){
        clearInterval(timer);
        clearInterval(warningTimer);
        timer = setInterval (refreshpage, refreshInterval);
    }
};
```

4. You are developing an e-commerce application that needs to integrate with an external payment gateway service.

step1: Generate client code

* Use a tool like apache Axis to wsimport
* provide the WSDL file URL & the target package name to the tool
* Tool will generate a set of Java classes that represent the service.

step2: Create a service client

* Create a instance of the service client class
* Set any required properties, such that the endpoint URL or authentication credentials.

Step3: Invoke the Service:

* Use the Service client instance to invoke desired operation.

* Pass any required parameters, such as payment details or order information.

* Handle the response from the which may include a transaction ID or error message.

Step4: Handle Errors:

* Catch any exceptions thrown by the Service invocation
* Handle error specific to the payment gateway Service, such as invalid payment detail.
* Implement the logic or fallback mechanisms if necessary