Application scope
==========
Write a code using applicationscope to print hit count of the application

```
<%
            Integer count=(Integer)application.getAttribute("hitcount");
            if(count == null)
                  count = 1;
            else
                  count++;

            application.setAttribute("hitcount", count);
      %>
<h1 style = 'color:red;'>Hit count of the application is :: <%=count %></h1>
```

Write a code using application scope to count no of users login to the application
     users => track through session.

```
<%@ page  session="true"%>
<%
      Integer count = (Integer) application.getAttribute("usercount");
      if (session.isNew()) {
            if (count == null)
                  count = 1;
            else
                  count++;
      }
      application.setAttribute("usercount", count);
%>
<h1 style='color: red;'>Hit count of the application is ::<%=count%></h1>
```

Write a code to display the no of requests in current session?
      session -> hold the data uniquely w.r.t user, so keep it in session object.

```
<%
      Integer count = (Integer) session.getAttribute("sessionRequestCount");
      if (count == null)
            count = 1;
      else
            count++;
      session.setAttribute("sessionRequestCount", count);
%>
<h1 style='color: red;'>Hit count of the application is ::<%=count%></h1>
```

Note: In all the above programs,initally the all the variables would not be available in the respective object, so null value will
          be returned, based on the condition the variable would be created with the respective values and stored back in
          the respective objects.

To retrieve the value from the PageContext object w.r.t to the scope we need to use the following methods
      a. pageContext.getAttribute(String name, int scope);

Scope levels
==========
PAGE_SCOPE = 1
REQUEST_SCOPE = 2
SESSION_SCOPE = 3
APPLICATION_SCOPE = 4

Demonstrate the need of pageContext object

```
====================================
first.jsp
=====
<%
      pageContext.setAttribute("p", "page");
      request.setAttribute("r", "request");
      session.setAttribute("s", "session");
      application.setAttribute("a", "application");

      pageContext.forward("second.jsp");
%>

second.jsp
=========
Page Scope attribute :: <%= pageContext.getAttribute("p",1)%><br/>
Request Scope attribute ::<%= pageContext.getAttribute("r",2)%><br/>
Session Scope attribute :: <%= pageContext.getAttribute("s",3)%><br/>
Application Scope atribute :: <%= pageContext.getAttribute("a",4) %>

Output
Page Scope attribute :: null
Request Scope attribute :: request
Session Scope attribute :: session
Application Scope atribute :: application

usage of findAttribute(string name)
============================
<%
      pageContext.setAttribute("page", "page");
      request.setAttribute("request", "request");
      session.setAttribute("session", "session");
      application.setAttribute("application", "application");
%>
<h1>Find Attribute ::<%=pageContext.findAttribute("a")%></h1>


JSP Actions
=========
In JSP technology, using scripting elements we are able to provide java code inside
jsp pages.
As per the theme of JSP writing java code is not allowed.
=> To eleminate java code from jsp pages we need to use "JSP Actions".
=> In JSP actions we provide Scripting tag in jsp page and we provide java code
w.rt. Scripting tag.

Note:
Whenever container encounters the Scripting tag, then container will execute
respective code by this an action will be
performed which  is called as "JSP Actions".

In JSP we have 2 types of Actions
      a. Standard Actions(supplied by jsp technology only)
      b. Custom Actions(as per the user needs by taking the support of SRS we can
define our own)

Standard Actions
==============
1. <jsp:useBean>
2. <jsp:setProperty>
```

3. <jsp:getProperty>
4. <jsp:include>
5. <jsp:forward>
6. <jsp:scriptlet>
7. <jsp:expression>
8. <jsp:delcaration>

What is java bean?
   It is a normal java class with setters, getters defined for private variables of a class.
   To promote serialziation for a java bean we use an interface called "Serializable".
   It is also called as "POJO".

Standard Actions
===============
```
<jsp:useBean id = "name of the reference "  scope="[scopes of jsp]"
                        class="name of the class for which object should be created"/>
      X idvalue=(X)Class.forName([supplied value in class]).newInstance();


<jsp:setProperty property ="" name = "" value = ""/>
            name.setPropertyValue(value supplied);
<jsp:getProperty property="" name = "" />
            name.getPropertyValue()

eg:
<jsp:useBean id="student" class="in.ineuron.bean.Student" scope="page">
      <jsp:setProperty property="id" name="student" value="10" />
      <jsp:setProperty property="name" name="student" value="sachin" />
      <jsp:setProperty property="address" name="student" value="MI" />
      <jsp:setProperty property="age" name="student" value="49" />
</jsp:useBean>
<jsp:getProperty property="id" name="student"/>

input.html
--------------
<table>
                  <tr>
                              <th>ID</th>
                              <td><input type='text' name='id' /></td>
                  </tr>
                  <tr>
                              <th>NAME</th>
                              <td><input type='text' name='name' /></td>
                  </tr>
                  <tr>
                              <th>AGE</th>
                              <td><input type='text' name='age' /></td>
                  </tr>
                  <tr>
                              <th>ADDRESS</th>
                              <td><input type='text' name='address' /></td>
                  </tr>
                  <tr>
                              <th></th>
                              <td><input type='submit' value='reg' /></td>
                  </tr>
```

```
</table>

As notice above inside html page name attribute values and in the bean instance
variable(fieldname) are same
      <input type ='text' name = 'id'/>
      <input type ='text' name = 'name'/>
      <input type ='text' name = 'age'/>
      <input type ='text' name = 'address'/>

class Student
{
            Integer id;
            String name;
            Integer age;
            String address;
}
If both variable names are same, then instead of binding the value to each variable
explicitly  as shown below
<jsp:setProperty property="id" name="student" value='<%=id%>' />
<jsp:setProperty property="name" name="student" value='<%=name%>' />
<jsp:setProperty property="address" name="student" value='<%=address%>' />
<jsp:setProperty property="age" name="student" value='<%=age%>' />

we can bind it automatically using "*"
<jsp:setProperty property="*" name="student" />

2. <jsp:include>=> include request Dispatching mechanism.
    <jsp:forward>=> forward request Dispatching mechanism.
    <jsp:param name = '' value=''">  => To add new values to request object and
send it to the respective page we use
                                                          <jsp:param name = ''
value=''/>.

3. deprecated jsp actions are
      <jsp:plugin>
      <jsp:fallback>
      <jsp:params>

Customactions
============
    These are the actions which are developed by developers as per their
application requirements.
    In jsp two types of tags are available
            a. standardactions -> predefined tags known to container
            b. customactions   -> inform explicitly to the container.

To prepare custom tags in Jsp pages we use the following syntax.
      <prefix_name:tag_name>
                  ;;;;;;
                  ;;;;;;
                  ;;;;;;
                  ;;;;;;
      </prefix_name:tag_name>
If we want to design custom tags in our jsp application, then we use the following
3 elements
      a. jsp page with taglib directive
      b. TLD file(Tag library descriptor)
      c. Tag Handler class.
```