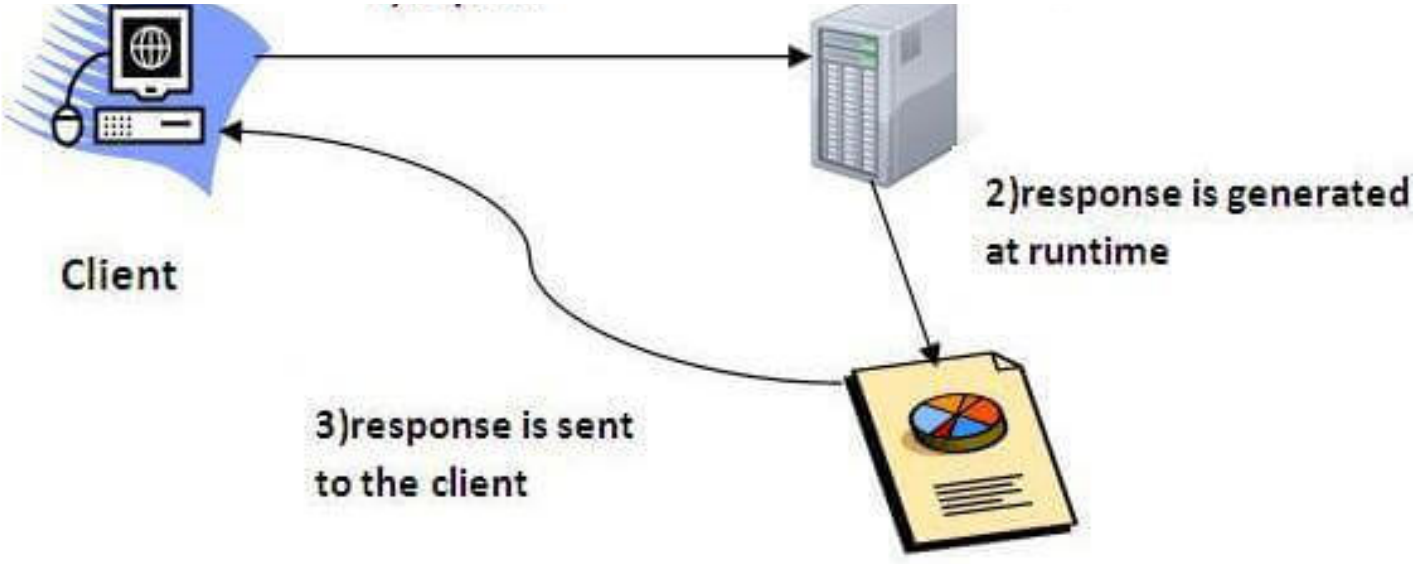
13: Servlets & JSP

Cuprins

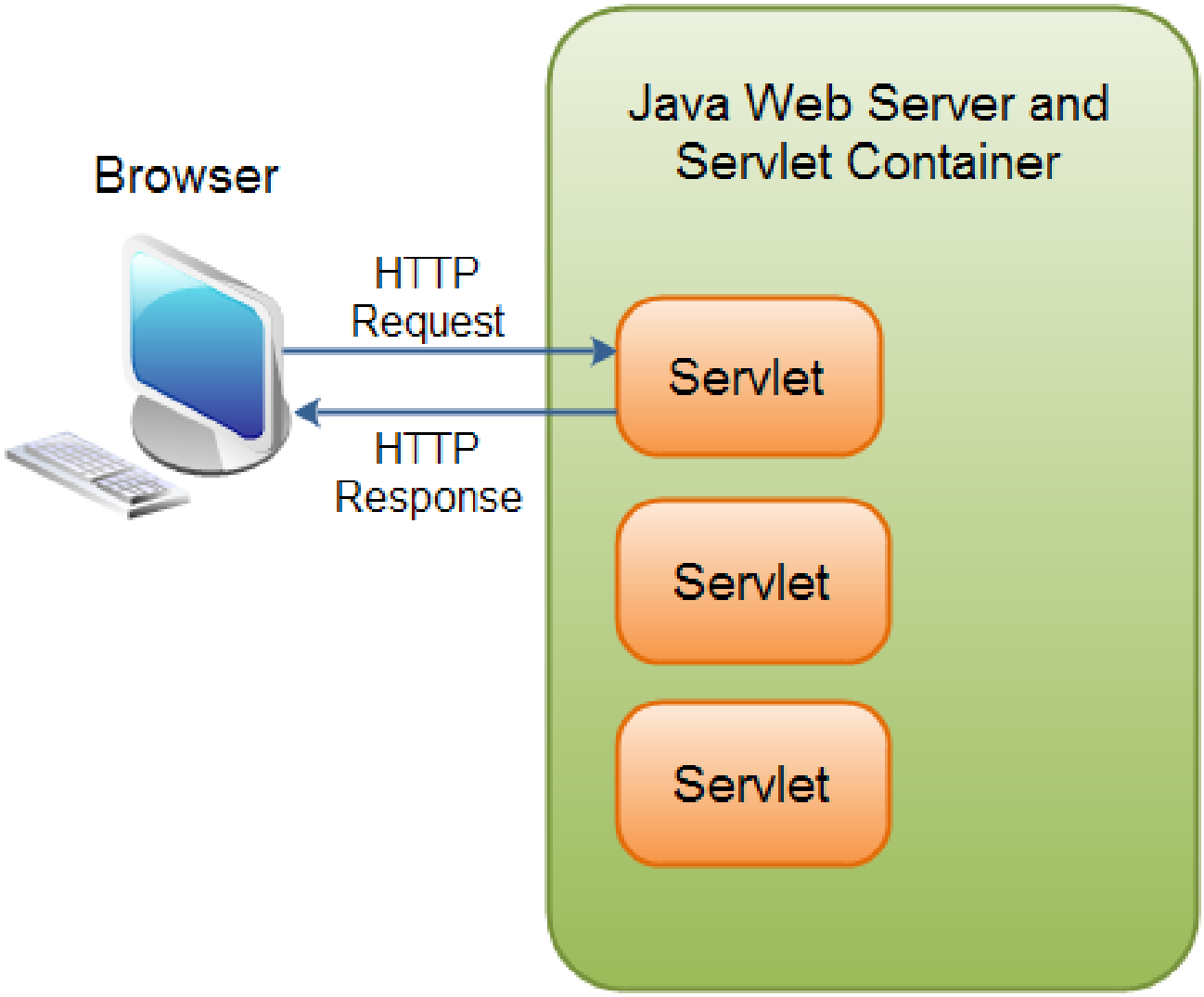
1. Introducere Servlet-uri



- O pagină Web poate fi statică sau dinamică.
- O pagină statică este furnizată utilizatorilor de către server exact în forma în care este salvată. Astfel, conținutul său este același pentru toți utilizatorii, în orice context.
- În schimb, conținutul unei pagini dinamice este generat de către o aplicație Web aflată pe server ori de câte ori pagina este vizualizată de către un utilizator, respectiv la efectuarea unei cereri HTTP pentru o pagină dinamică aplicația Web va genera conținutul său chiar în acel moment, ci nu va prelua direct un conținut salvat anterior.
- Astfel, o pagină dinamică este utilizată pentru a furniza utilizatorului un conținut particularizat conform unor parametrii pe care acesta i-a transmis server-ului, folosind, de obicei, un formular HTML (de exemplu, funcția de căutare a unui anumit produs, funcția de autentificare etc.).

- Servlet-urile sunt aplicații Java gestionate și executate de un server Web specializat, precum Apache Tomcat, GlassFish etc.
- Cu ajutorul unui servlet se pot dezvolta aplicații care implementează paradigma cerere/răspuns.
- Practic, un servlet este o clasă scrisă în limbajul Java al cărei scop este generarea dinamică de date într-un server HTTP.
- O astfel de clasă poate crea atât conținut HTML, cât și documente XML, PDF, imagini, fișiere etc.
- Totuși, de obicei, un servlet este folosit împreună cu protocolul HTTP pentru a genera pagini web dinamice.

- În general, o aplicație servlet are un rol similar unei aplicații CGI (Common Geteway Interface), însă oferă avantaje suplimentare, cum ar fi:
  1. o performanță mai bună, deoarece se creează pentru fiecare client câte un fir de executare, ci nu câte un proces;
  2. executarea într-un context Web, deci nu este necesară crearea mai multor procese pentru a prelua cereri de la mai mulți clienți;
  3. independența de platformă, deoarece un servlet este scris în limbajul Java;
  4. asigurarea unui nivel înalt de securitate al resurselor disponibile pe server (fișiere, baze de date etc.);
  5. asigurarea unei comunicări transparente cu aplicații de tip applet, cu sisteme de gestiune a bazelor de date sau cu alte tipuri aplicații, folosind atât comunicarea prin socket-uri, cât și mecanismul RMI (Remote Method Invocation);
  6. posibilitatea de a utiliza toate facilitățile Java prin intermediul pachetelor standard.



- Un servlet nu este rulat direct de către server, ci de către un container web.
- Acesta este o componentă software a unui server care are rolul de a gestiona aplicații Java de tip server, precum servlet, Java Server Pages (JSP), Java Server Faces (JSF) etc.
- Sunt disponibile mai multe containere Web, cele mai populare fiind Apache Tomcat, Oracle GlassFish etc.

- În concluzie, un container web gestionează ciclul de viață al unui servlet, asociază un URL cu un anumit servlet, asigură securitatea servlet-ului etc.

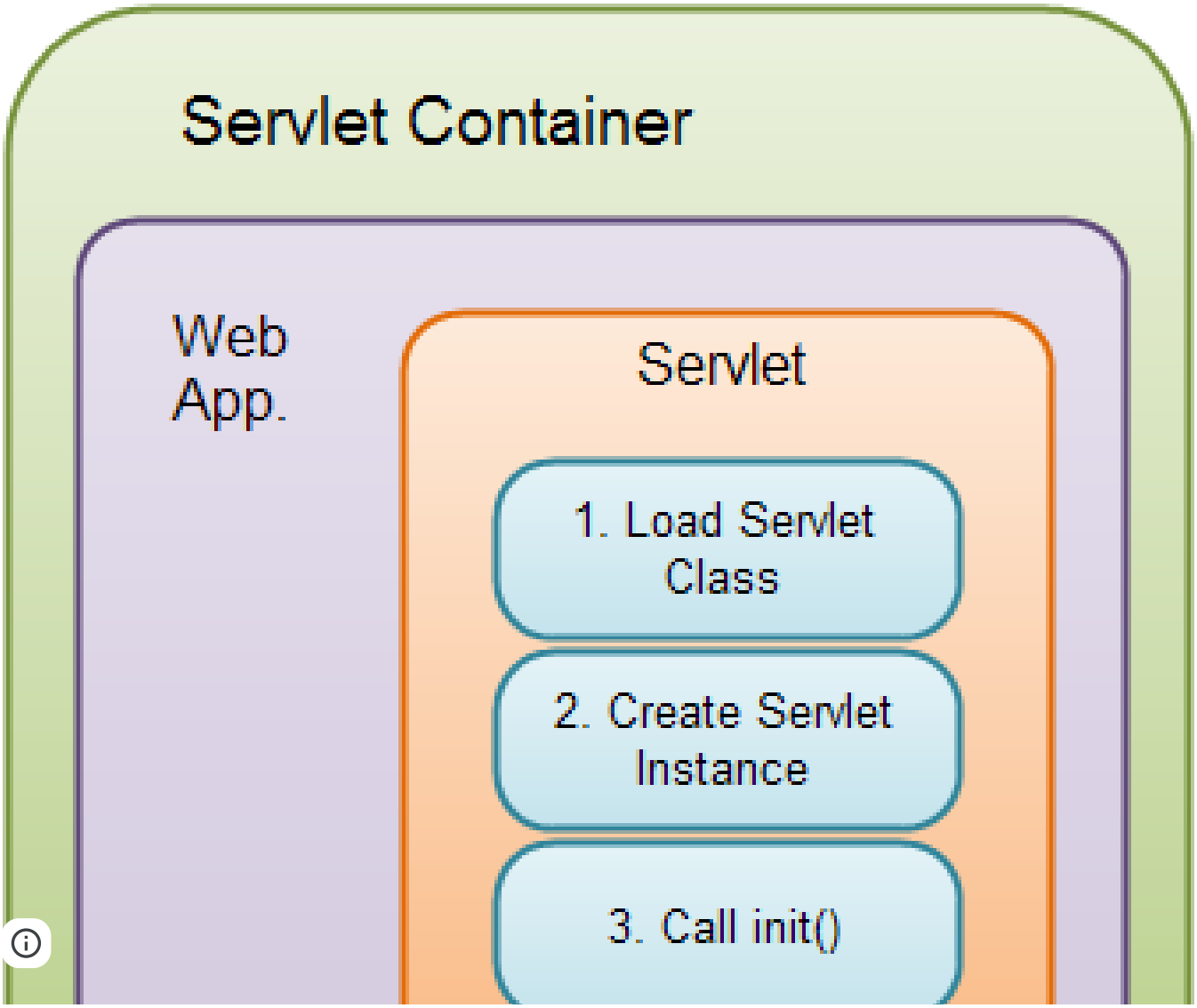
- Practic, comunicarea dintre un client (de obicei un browser) și un servlet se realizează astfel:
  1. pentru a răspunde unei cereri de la un client către un anumit servlet, serverul trimite mai departe cererea către container, care are rolul de a instanția obiectului respectiv și de a apela metodele necesare;
  2. serverul Web, după executarea servlet-ului, trimite înapoi browser-ului un fișier HTML, pe care acesta îl afișează în mod corespunzător.

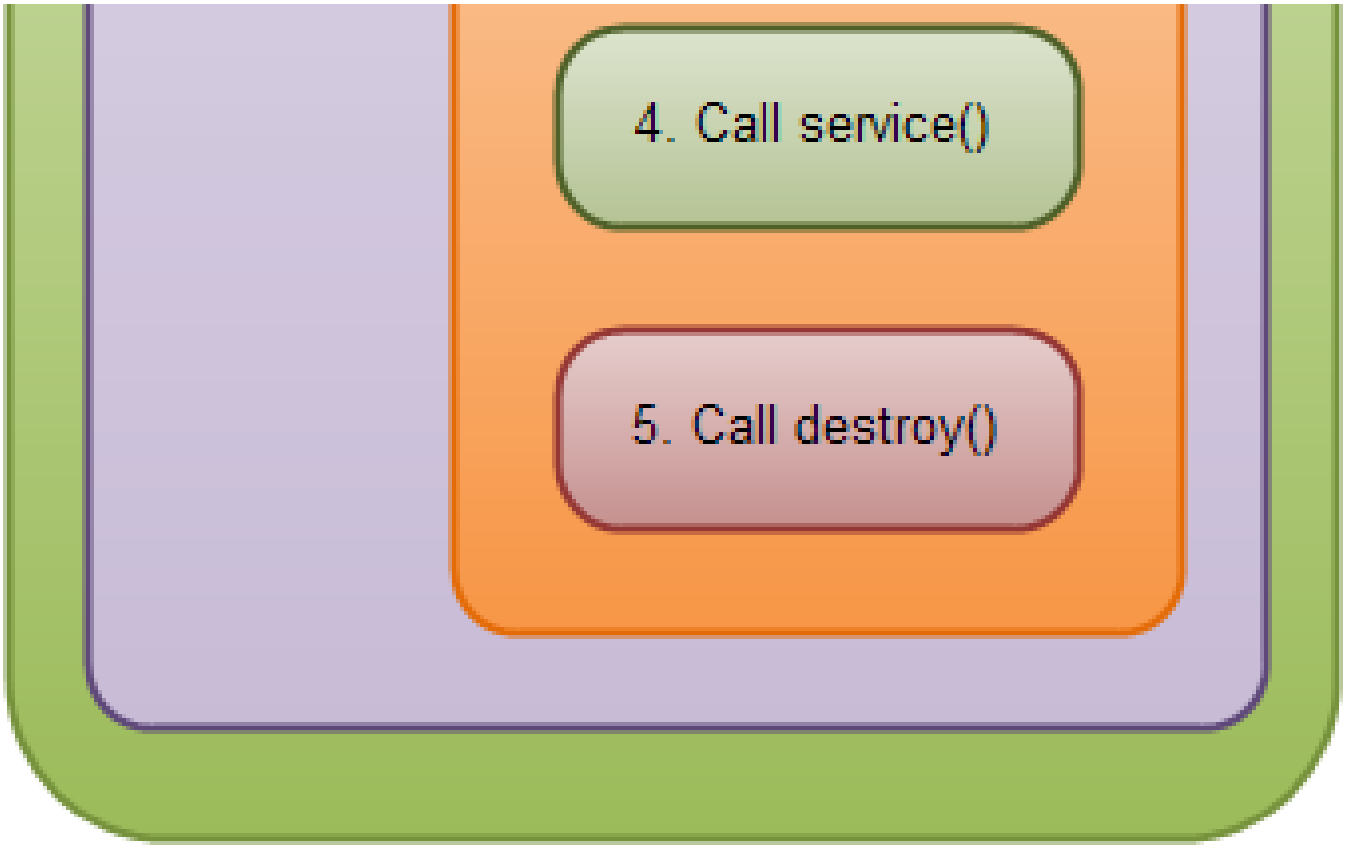
1. returnează un document HTML construit dinamic pe baza cererii unui client;
2. procesează datele completate de utilizatori printr-un formular HTML și returnează un răspuns;
3. furnizează suport pentru autentificarea utilizatorului, precum și utilizarea unor mecanisme de securitate;
4. interacționează cu resursele server-ului, cum ar fi baze de date, fișiere etc.;
5. procesează intrările de la mai mulți clienți pentru aplicații, fiecare pe un fir de executare separat, cum ar fi, de exemplu, jocurile în rețea;
6. redirecționează cereri de la un servlet la altul.

- Definirea unui servlet se poate realiza prin extinderea clasei abstracte **HttpServlet** din pachetul javax.servlet, care conține următoarele metode:

1. metoda **init**() se apelează o singură dată, atunci când este încărcat servlet-ul (similar unui constructor). Practic, servlet-ul este creat în momentul în care un client emite o cerere, dar se poate opta și pentru a-l crea în momentul pornirii server-ului;
2. metoda **service**() este automat apelată ca răspuns la cererea fiecărui client și poate fi suprascrisă pentru a furniza o funcționalitate implicită (servlet-urile care extind **HttpServlet** pot să nu suprascrie această metodă);
3. metoda **destroy**() este apelată când servlet-ul este oprit de către server-ul Web.

3. Ciclul de viață al unui Servlet





- Ciclul de viață al unui servlet este următorul:
  1. serverul încarcă servlet-ul când acesta este cerut de către client sau la pornirea server-ului;
  2. serverul creează o instanță a clasei servlet-ului pentru deservirea tuturor clienților, pentru fiecare client fiind alocat automat un fir de executare separat;
  3. serverul apelează metoda **init**() a servlet-ului;
     În momentul primirii unei cereri pentru servlet, serverul instanțiază:
     - un obiect HttpServletRequest folosind datele incluse în cererea clientului
     - un obiect HttpServletResponse care furnizează metode pentru returnarea răspunsului
  4. servlet-ul apelează metoda **service**(), astfel:
     - metoda service() primește ca parametrii obiectele construite la pasul anterior și, la rândul său, apelează metodele specifice protocolului HTTP pentru transmiterea datelor de la/către client, respectiv metodele doGet() sau doPost();
     - metoda service() procesează cererea clientului prin intermediul obiectului de tip HttpServletRequest și furnizează un răspuns prin obiectul HttpServletResponse;
  5. se apelează metoda **destroy**() în cazul în care containerul de servlet-uri al server-ului Web inițiază oprirea sa.

În afara metodelor prezentate mai sus, în clasa HttpServlet mai sunt definite și alte metode specifice: https://www.javaguides.net/2019/02/httpservlet-class-example-tutorial.html.

4. Rularea unui Servlet

- De exemplu, pentru a configura serverul Apache Tomcat trebuie să urmați pașii prezentați în pagina https://www.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_HowTo.html.
- O metodă mai simplă de implementare și rulare a unui servlet o constituie utilizarea unui mediu integrat de dezvoltare (IDE), astfel:
  - Netbeans IDE 8.2 - https://www.studytonight.com/servlet/creating-servlet-in-netbeans.php
  - IntelliJ IDEA - https://www.heavyweightsoftware.com/writing-a-basic-servlet-with-intellij-idea/
  - Eclipse IDE - https://beginnersbook.com/2017/07/how-to-create-and-run-servlet-in-eclipse-ide/

Exemplul 1: un servlet care afișează un număr aleatoriu într-o pagină Web

```
public class Test extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Suma</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1 align=center>"+this.getServletInfo()+"</h1>");
            out.println("</br>");
            Random PRNG = new Random();
            out.println("<h1 align=center>Numărul aleatoriu: " + PRNG.nextInt() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
    @Override
    public String getServletInfo() {
        return "Servlet care afișează un număr aleatoriu!";
    }
}
```

- Caracterul dinamic al paginii generate se poate testa reîncărcând pagina de mai multe ori și observând faptul că numărul aleatoriu afișat se schimbă de fiecare dată!

5. Transmiterea datelor către un Servlet

- Metoda **GET** transmite informațiile către server în clar, sub forma unui șir de caractere atașat link-ului:

  http://www.test.com?key1=value1&key2=value2

- Datele transmise prin metoda **GET** sunt restricționate la maxim 1024 caractere și nu pot conține date în format binar. Deoarece valorile parametrilor sunt vizibile în link, nu se poate folosi metoda **GET** pentru a transmite date confidențiale! Mai mult, datele transmise folosind metoda GET rămân în cache-ul browser-ului și sunt salvate în istoricul navigării (History)!

- Spre deosebire de metoda **GET**, metoda **POST** transmite informații către server prin intermediul header-ului HTTP. Astfel, nu există restricții referitoare la dimensiunea datelor trimise, iar datele pot fi și în format binar.

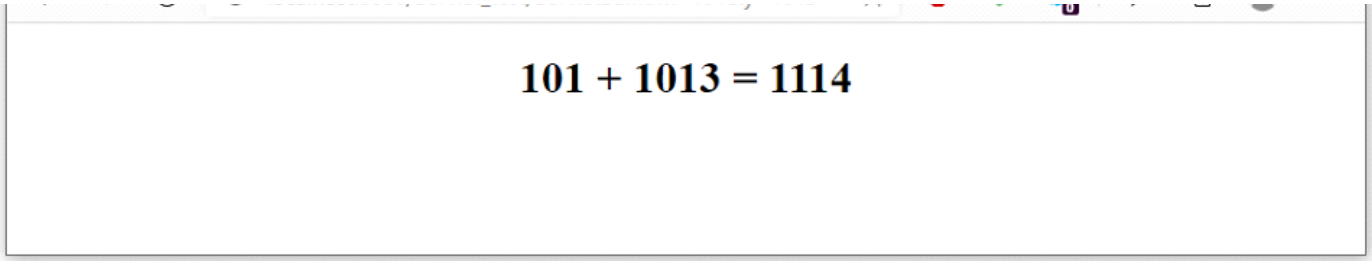
- Pentru o cerere HTTP de tip **GET**, în cadrul servlet-ului se apelează metoda **doGet**(), respectiv **doPost**() pentru o cerere de tip **POST**. Ambele metode au ca parametrii un obiect de tip **HttpServletRequest** care conține informații despre cererea clientului și un obiect de tip **HttpServletResponse** în care se va scrie răspunsul servlet-ului. Atenție, dacă servlet-ul nu implementează metoda corespunzătoare metodei HTTP utilizate pentru transmiterea datelor se va genera o eroare!

- În concluzie, parametrii unei cereri fie sunt incluși în URL, fie sunt încapsulați în corpul unei cereri HTTP. Indiferent de varianta utilizată, valorile parametrilor pot fi preluate în cadrul celor două metode, sub forma unor șiruri de caractere, folosind metoda String **getParameter**(String nume_parametru).

Exemplul 2: un servlet care afișează suma a două numere transmise folosind metoda GET

```
public class ServletSuma extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Suma</title>");
            out.println("</head>");
            out.println("<body>");
            int x = Integer.parseInt(request.getParameter("x"));
            int y = Integer.parseInt(request.getParameter("y"));
            out.println("<h1 align=center>"+x+" + "+y+" = "+(x+y)+"</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

Servlet-ul va fi apelat printr-un link de forma
http://localhost:8080/Servlet_test/ServletSuma?x=101&y=1013
particularizat conform setărilor server-ului Web utilizat:



6. Transmiterea datelor prin formulare HTML

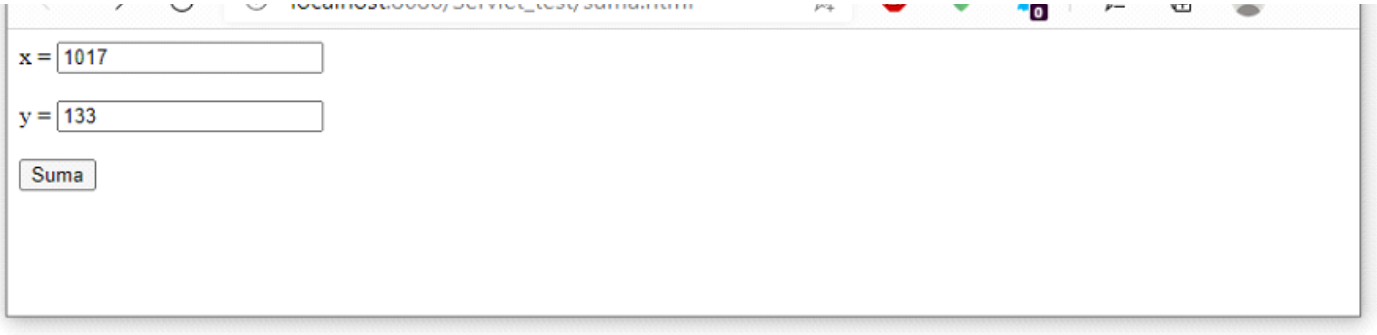
- Transmiterea datelor către un servlet se realizează, de obicei, prin intermediul formularelor HTML.
- Un formular se definește prin tag-ul **form**:

```
<form action="URL servlet" method="GET sau POST">
    componente formular
</form>
```

- În cadrul unui formular se definesc componente grafice specifice, cum ar fi:
  1. etichetă: <label> text </label>
  2. câmp de text: <input type="text" name="nume parametru">
  3. buton: <button type="button sau submit sau reset"> text </button>
- Mai multe detalii despre formularele HTML se pot găsi, de exemplu, în următoarea pagină: https://www.tutorialspoint.com/html/html_forms.htm.
- **Exemplul 3:** un servlet care afișează suma a două numere introduse într-un formular HTML și transmise folosind metoda POST
- Formularul poate fi definit într-un fișier HTML, de exemplu în fișierul suma.html, astfel:

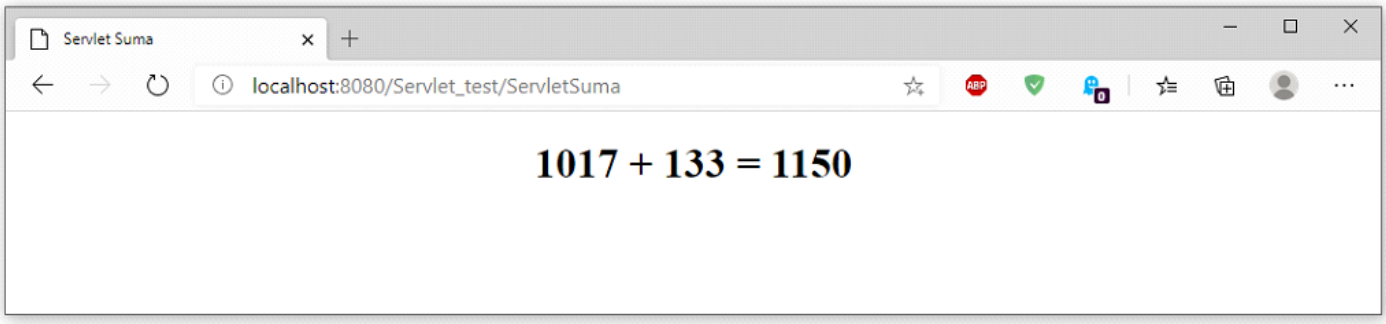
```
<html>
   <head>
      <title>Formular servlet suma</title>
   </head>
   <body>
      <form action = "http://localhost:8080/Servlet_test/ServletSuma" method = "POST">
         <label>x = </label><input type = "text" name = "x">
         <br/>
         <br/>
         <label>y = </label><input type = "text" name = "y">
         <br/>
         <br/>
         <button type="submit">Suma</button>
      </form>
   </body>
</html>
```

- Evident, URL-ul servlet-ului http://localhost:8080/Servlet_test/ServletSuma indicat în parametrul action al formularului trebuie particularizat în funcție de setările server-ului Web utilizat!



- Codul sursă al acestui servlet este aproape identic cu cel al servlet-ului prezentat anterior, care folosea metoda GET pentru transmiterea parametrilor, singura diferență constând în faptul că se va implementa metoda corespunzătoare **doPost**():

```
public class ServletSuma extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Suma</title>");
            out.println("</head>");
            out.println("<body>");
            int x = Integer.parseInt(request.getParameter("x"));
            int y = Integer.parseInt(request.getParameter("y"));
            out.println("<h1 align=center>"+x+" + "+y+" = "+(x+y)+"</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```



- Analizând link-ul paginii generate de servlet ca răspuns al cererii de tip **POST**, se observă faptul că el nu mai conține valorile parametrilor x și y, așa cum se întâmpla în cazul cererii de tip **GET**!

Exemplul 4: un servlet care afișează angajații unei firme care au salariul cel puțin egal cu o valoare min dată

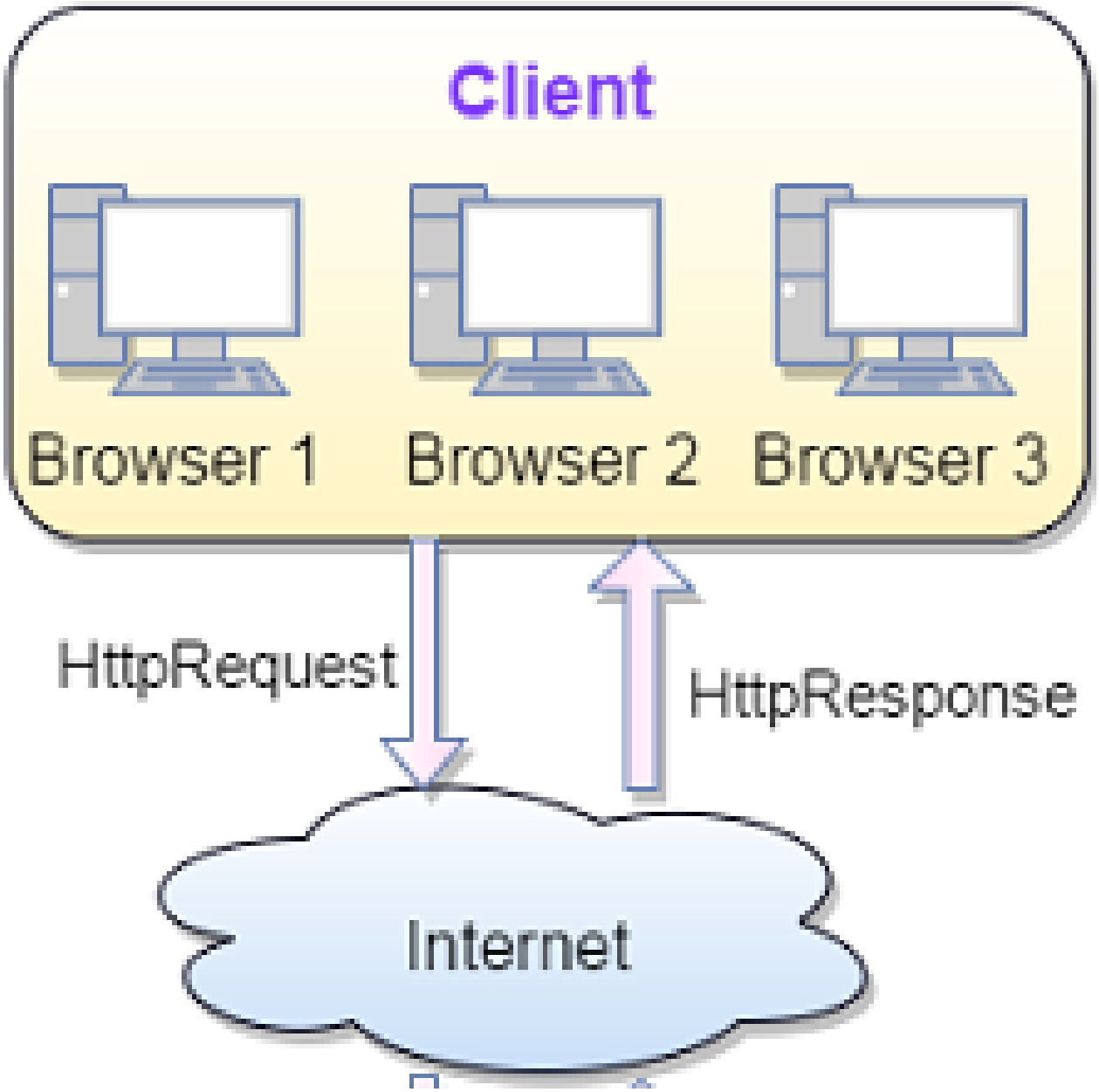
- Vom presupune faptul că informațiile despre angajați sunt păstrate în baza de date AngajațiDB, într-o tabelă numită Angajați, având câmpurile Nume (VARCHAR), Vârsta (INTEGER) și Salariu (DOUBLE).
- codul sursă de mai jos, baza de date este stocată folosind SGBD-ul Apache Derby și are URL-ul jdbc:derby://localhost:1527/AngajatiDB.

- Metoda **init**() a servlet-ului fost redefinită astfel încât conectarea la baza de date să se realizeze în momentul în care serverul Web încarcă servlet-ul, iar metoda **destroy**() a fost redefinită astfel încât conexiunea cu baza de date să fie închisă în momentul opririi servlet-ului de către serverul Web.

```
public class SalariiAngajati extends HttpServlet {
    Connection conn = null;

    @Override
    public void init() throws ServletException {
        try {
            conn = DriverManager.getConnection(
            "jdbc:derby://localhost:1527/AngajatiDB","Popescu Ion", "12345");
        } catch (SQLException ex) { System.err.println("Eroare: " + ex); }
    }

    @Override
    public void destroy() {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException ex){System.err.println("Eroare: " + ex);}
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<body>");

            float min = Float.parseFloat(request.getParameter("min"));

            PreparedStatement pst = null;
            ResultSet rs = null;


            try {
                pst = conn.prepareStatement("SELECT * FROM Angajati WHERE Salariu > ?");
                pst.setFloat(1, min);

                rs = pst.executeQuery();

                if (!rs.next())
                    out.println("<h1>Niciun angajat nu are salariul cel putin" + min + " RON!</h1>");
                else {
                    out.println("<h1>Angajatii avand salariul minim " + min +
                    " RON:</h1>");
                    out.println("</br>");
                    while (rs.next())
                        out.println("<h2>" + rs.getString("Nume") + " - " +
                    rs.getFloat("Salariu") + " RON - " +
                    rs.getInt("Varsta") + " ani</h2>");
                }
            } catch(SQLException ex){
                System.err.println("Eroare: " + ex);
            } finally {
                try {
                    if (rs != null) rs.close();
                    if (pst != null)pst.close();
```

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

7. Java Server Pages



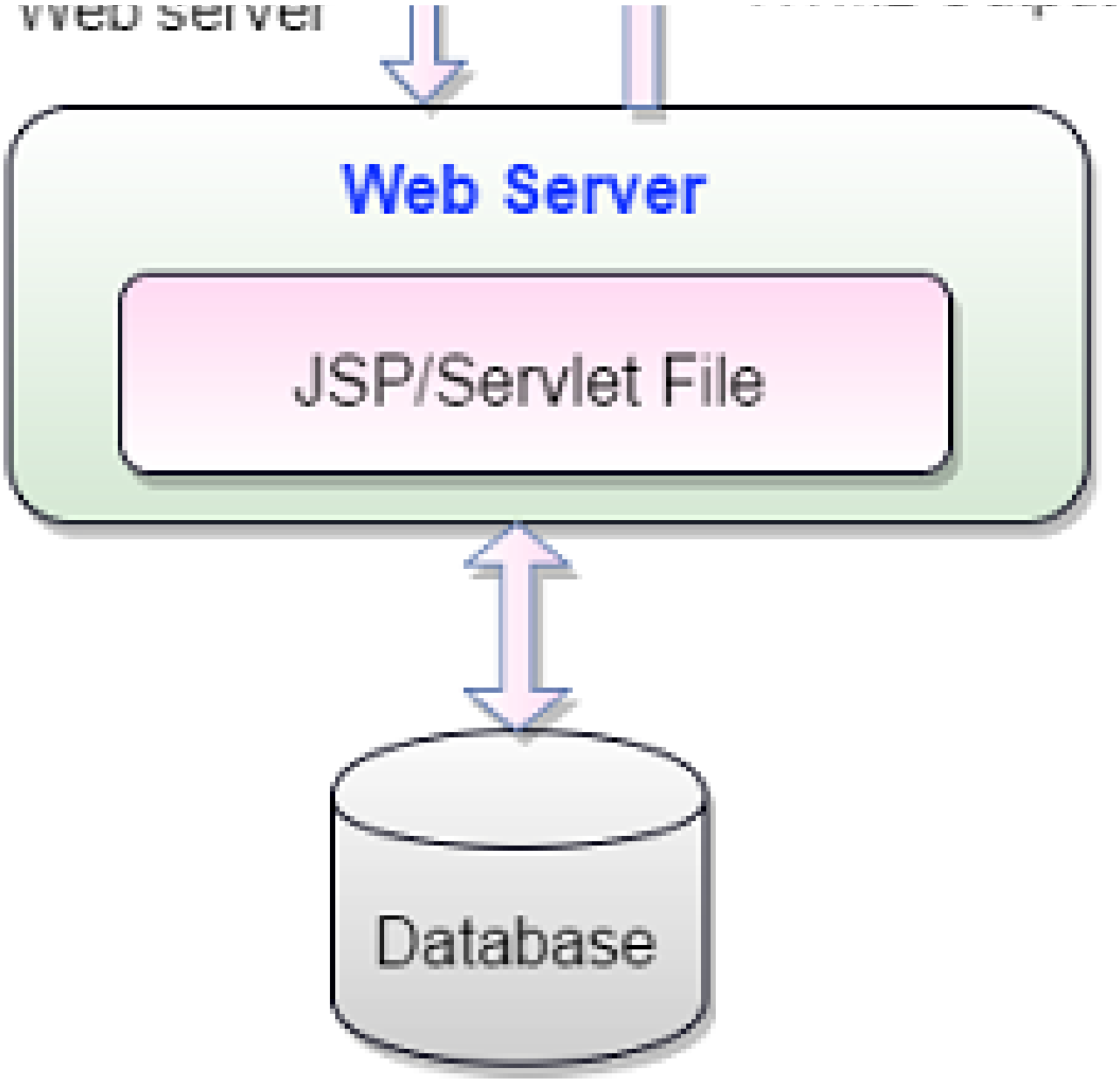


Fig: JSP Architecture

- Spre deosebire de un servlet, care generează o pagină Web cu un conținut dinamic prin cod Java, tehnologia JSP extinde limbajul HTML oferind posibilitatea de a însera cod Java sub forma unor scripturi.
- Astfel, paginile Web sunt generate dinamic prin conversia unor fișiere de script în module Java executabile.
- Practic, o pagină JSP are structura unui fișier HTML, dar cu extensia .jsp.

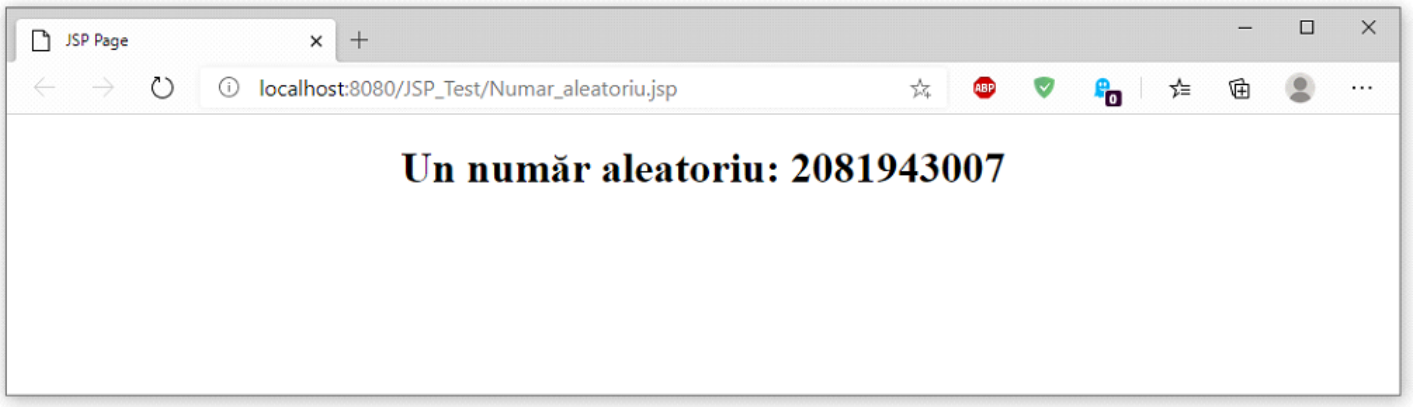
- Un avantaj important al unei pagini JSP față de un servlet este dat de faptul că are loc o separare clară a conținutului HTML static față de cel dinamic, astfel orice modificare referitoare la estetica paginii nu conduce la recompilare, așa cum se întâmplă în cazul unui servlet.

- Codul Java (scriptlet) se inserează într-o pagină HTML folosind tag-uri dedicate, cele mai utilizate fiind următoarele:

- <!--comentariu HTML -->
- <%--comentariu JSP --%>
- <%! declarare Java %>
- <%= expresie Java %>
- <% cod Java %>

Exemplul 5: o pagină JSP care afișează un număr aleatoriu (echivalentă cu servlet-ul din Exemplul 1)

```
<%@page import="java.util.Random"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Număr aleatoriu JSP</title>
</head>
<body>
<%! Random PRNG = new Random(); %>
<h1 align="center">
Un număr aleatoriu: <%= PRNG.nextInt() %>
</h1>
</body>
</html>
```



8. Ciclul de viață al unei pagini JSP

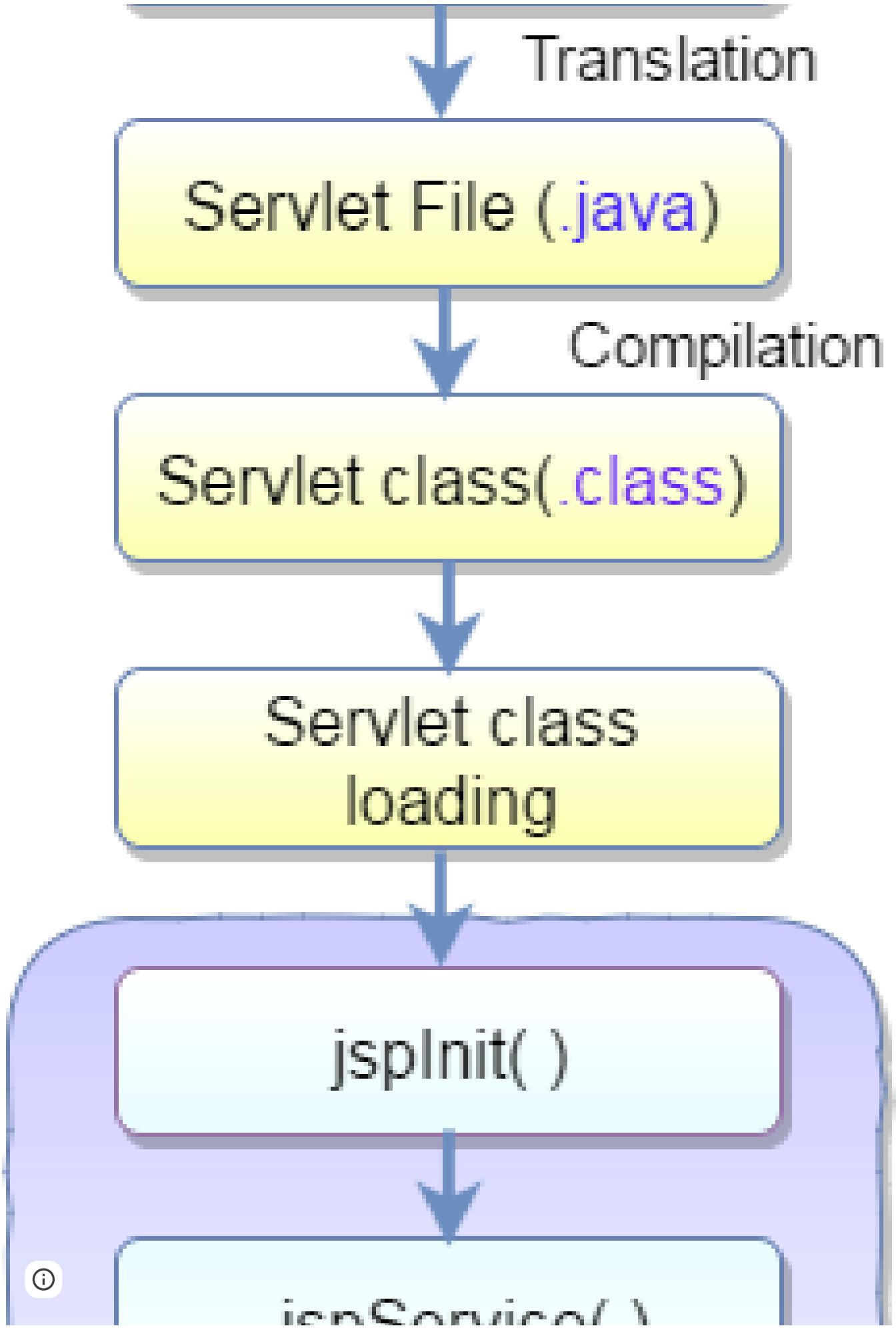

Translation
Servlet File (.java)
Compilation
Servlet class(.class)
Servlet class loading
jspInit( )

Fig: JSP Life Cycle

- Ciclul de viață al unei pagini JSP este următorul (https://www.tutorialride.com/jsp/java-server-pages-jsp-tutorial.htm):

1. **translatarea** presupune transformarea paginii JSP (.jsp) într-un servlet (.java);
2. **compilarea** presupune generarea fișierului .class corespunzător servlet-ului, astfel:
   - în momentul în care serverul Web primește o cerere pentru o pagină JSP, motorul JSP verifică dacă este necesară compilarea sa, respectiv dacă nu a mai fost compilată sau dacă a fost modificată de la ultima compilare;
   - compilarea presupune parsarea paginii JSP, translatarea sa într-un servlet și compilarea servlet-ului;
3. **încărcarea** presupune salvarea în memorie a fișierul .class corespunzător servlet-ului;
4. **instanțierea** presupune instanțierea unui obiect servlet de către containerul Web
5. **inițializarea** presupune apelarea metodei jspInit() a servlet-ului;
6. **procesarea** cererilor presupune crearea, pentru fiecare client în parte, a unui un fir de executare, apelarea metodei **_jspService**() și generarea unui răspuns HTML;
7. **distrugerea** presupune eliberarea zonei de memorie alocată servlet-ului și se realizează prin apelarea metodei jspDestroy().

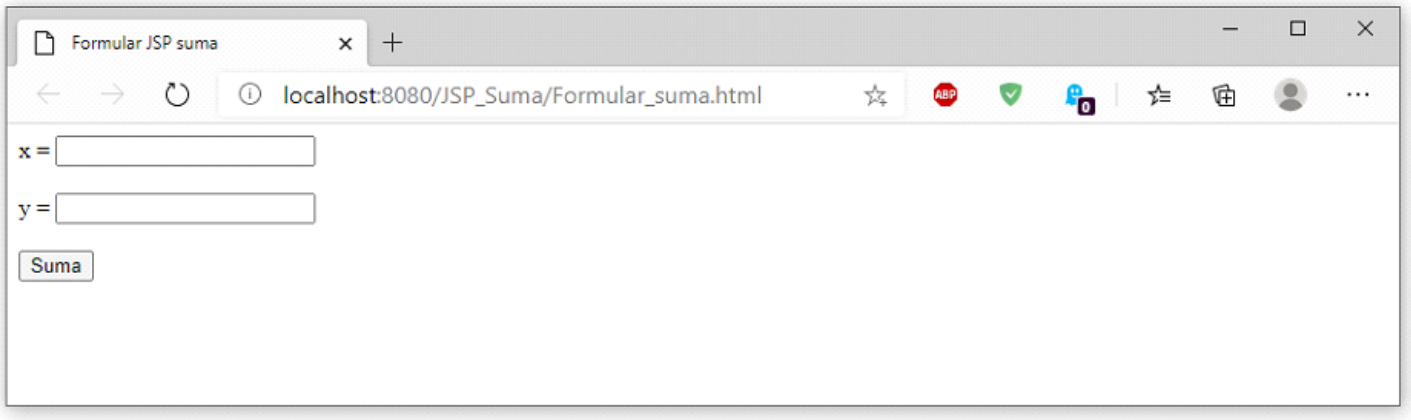
1. **request** – obiect de tip HttpServletRequest asociat paginii, echivalent cu primul parametru al metodelor doGet() și doPost() dintr-un servlet;
2. **response** – obiect de tip HttpServletResponse asociat paginii, echivalent cu al doilea parametru al metodelor doGet() și doPost() dintr-un servlet;
3. **out** – obiect de tip PrintWriter folosit pentru generarea răspunsului către client;
4. **page** – sinonim pentru referința this;
5. **Exception** – obiect de tip Exception asociat paginii JSP.

Exemplul 6: o pagină JSP care afișează suma a două numere introduse într-un formular HTML și transmise folosind metoda POST (echivalentă cu servlet-ul din Exemplul 2)

- Formularul poate fi definit într-un fișier HTML, de exemplu în fișierul Formular_suma.html, astfel:

```
<html>
<head>
<title>Formular JSP suma</title>
</head>
<body>
<form action = "http://localhost:8080/JSP_Suma/Suma.jsp" method = "POST">
<label>x = </label><input type = "text" name = "x"><br/><br/>
<label>y = </label><input type = "text" name = "y"><br/><br/>
<button type="submit">Suma</button>
</form>
</body>
</html>
```

- Evident, URL-ul paginii JSP http://localhost:8080/JSP_Suma/Suma.jsp indicat în parametrul action al formularului trebuie particularizat în funcție de setările server-ului Web utilizat!



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Suma</title>
</head>
<body>
<%
int x = Integer.parseInt(request.getParameter("x"));
int y = Integer.parseInt(request.getParameter("y"));
%>
<h1 align=center><%= x%> + <%= y%> = <%= x+y%></h1>
<br/>
<br/>
<h2>
<a href = "http://localhost:8080/JSP_Suma/Formular_suma.html">
Back
</a>
</h2>
</body>
</html>
```

- Se observă faptul că valorile parametrilor x și y au fost preluate din header-ul HTTP utilizând obiectul predefinit request!
- În afara obiectelor predefinite menționate anterior, unei pagini JSP îi este asociat, pe parcursul întregului său ciclu de viață, un obiect application.
- Practic, obiectul application este creat în momentul instanțierii servlet-ului asociat paginii JSP (apelarea metodei **jspInit**()) și este distrus în momentul distrugerii servlet-ului asociat (apelarea metodei **jspDestroy**()).
- Folosind acest obiect predefinit, se pot accesa valorile unor parametri impliciți de configurare din fișierul web.xml sau se pot crea și manipula parametri dedicați.
- Manipularea acestor parametri se realizează folosind următoarele două metode:
  1. application.**setAttribute**(String Key, Object Value);
  2. application.**getAttribute**(String Key);

```
<%@page import = "java.io.*,java.util.*" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<head>
<title>Numărare accesări</title>
</head>
<body>
<%
Integer contor = (Integer)application.getAttribute("contorAccesari");
if(contor == null || contor == 0) {
%>
<h1 align=center>Bine ați venit!</h1><br/>
<%
contor = 1;
} else {
%>
<h1 align=center>Bine ați revenit!</h1></br>
<%
contor++;
}
application.setAttribute("contorAccesari", contor);
%>
<center>
<h2 style="color: red">Numarul total de accesari: <%= contor%></h2>
</center>
</body>
</html>
```

- Mai multe detalii despre modalitățile de utilizare ale servlet-urilor și JSP-urilor puteți găsi în următoarele pagini:
  1. http://tutorials.jenkov.com/java-servlets/index.html
  2. https://www.tutorialspoint.com/servlets/index.htm
  3. http://www.ntu.edu.sg/home/ehchua/programming/java/javaservlets.html
  4. https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaServerPages.html
  5. https://www.tutorialspoint.com/jsp/index.htm