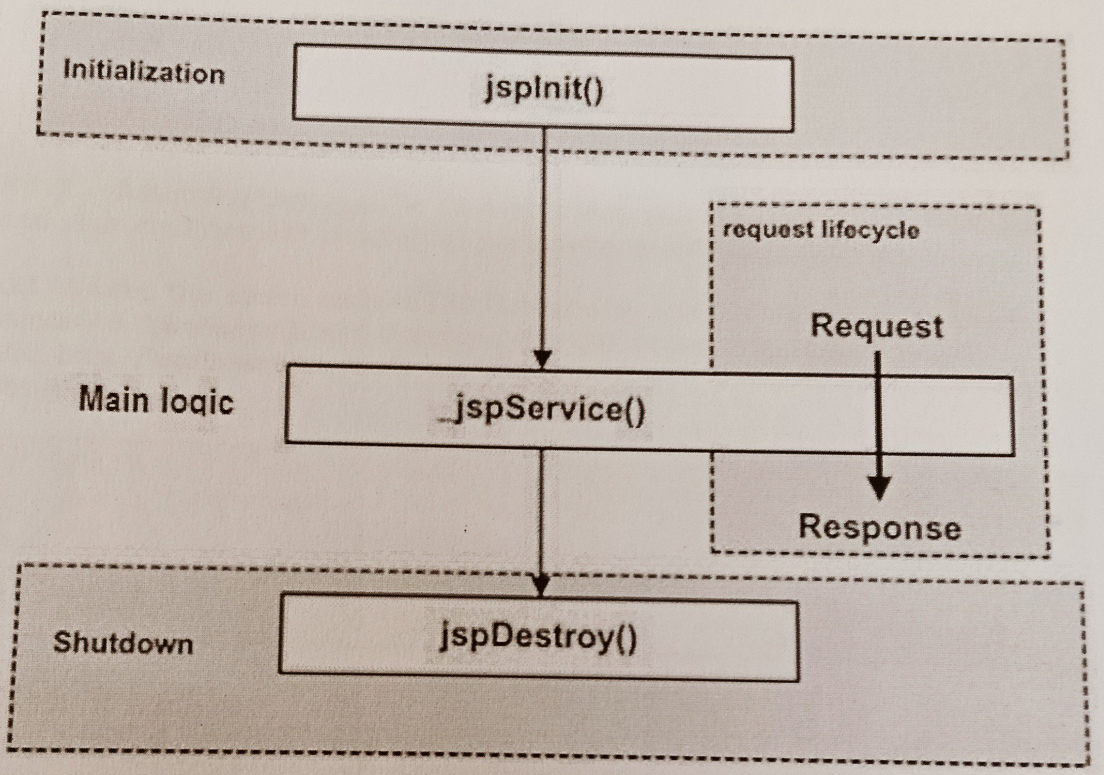
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

he four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



Elements of JSP

The elements of JSP have been described below –

The Scriptlet

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet –

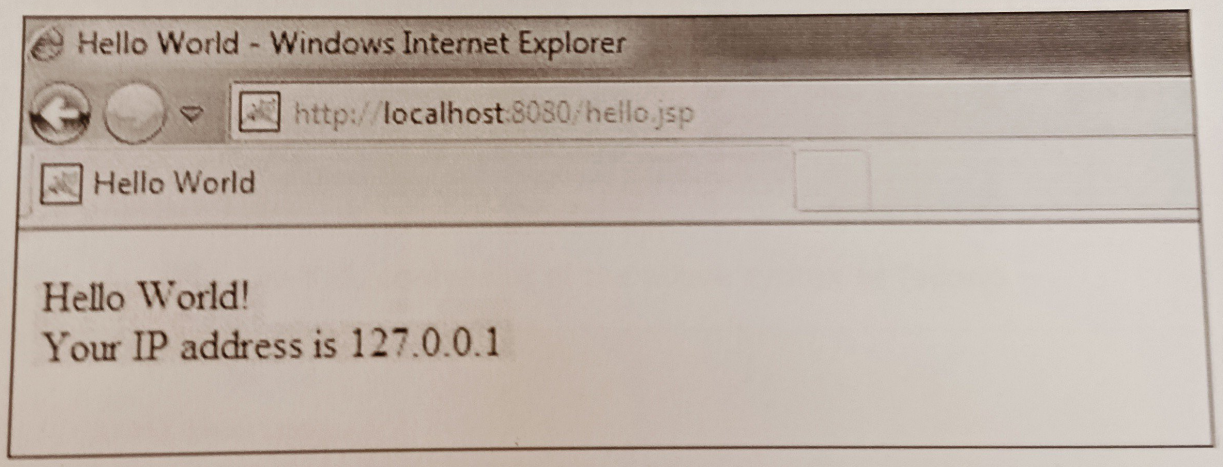
```
<% code fragment %>
```

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP −

```
<html>
   <head><title>Hello World</title></head>

   <body>
      Hello World!<br/>
      <%
         out.println("Your IP address is " +
request.getRemoteAddr());
      %>
   </body>
</html>
```

NOTE − Assuming that Apache Tomcat is installed in C:\apache-tomcat-7.0.2 and your environment is setup as per environment setup tutorial.

Let us keep the above code in JSP file **hello.jsp** and put this file in **C:\apache-tomcat7.0.2\webapps\ROOT** directory. Browse through the same using URL **http://localhost:8080/hello.jsp**. The above code will generate the following result −



JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declarations −

```
<%! declaration; [ declaration; ]+ ... %>
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:declaration>
```

code fragment
</jsp:declaration>

Following is an example for JSP Declarations –

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP Expression

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression –

```
<%= expression %>
```

You can write the XML equivalent of the above syntax as follows –

```
<jsp:expression>
   expression
</jsp:expression>
```

Following example shows a JSP Expression –

```
<html>
   <head><title>A Comment Test</title></head>

   <body>
      <p>Today's date: <%= (new
java.util.Date()).toLocaleString()%></p>
   </body>
</html>
```

The above code will generate the following result –

Today's date: 11-Sep-2010 21:24:25

Implicit Objects in JSP

These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called **pre-defined variables**.

S.No.	Object & Description
1	**request** This is the **HttpServletRequest** object associated with the request.
2	**response** This is the **HttpServletResponse** object associated with the response to the client.
3	**out** This is the **PrintWriter** object used to send output to the client.
4	**session** This is the **HttpSession** object associated with the request.
5	**application** This is the **ServletContext** object associated with the application context.
6	**config** This is the **ServletConfig** object associated with the page.
7	**pageContext** This encapsulates use of server-specific features like higher performance **JspWriters**.
8	**page** This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
9	**Exception** The **Exception** object allows the exception data to be accessed by designated JSP.

```
<%@ page language="java" contentType="text/html;
<!DOCTYPE html>

<html>
<head>
  <title>JSP Implicit Objects Example</title>
</head>
<body>
  <h2>Using JSP Implicit Objects</h2>
```

```
    <!-- Using request object -->
    <p><b>Client IP Address:</b> <%= request.getRemoteAddr() %></p>

    <!-- Using response object -->
    <%
      response.setContentType("text/html");
    %>

    <!-- Using out object -->
    <%
      out.println("<p><b>Using out object:</b> Hello, this is a JSP implicit object
example!</p>");
    %>

    <!-- Using session object -->
    <%
      session.setAttribute("user", "John Doe");
    %>
    <p><b>Session Attribute (user):</b> <%= session.getAttribute("user") %></p>

    <!-- Using application object -->
    <p><b>Server Info:</b> <%= application.getServerInfo() %></p>

</body>
</html>
```

Explanation:

1. `request.getRemoteAddr()` → Fetches the client IP address.
2. `response.setContentType("text/html")` → Ensures the response content type is HTML.
3. `out.println()` → Prints a message to the client.
4. `session.setAttribute("user", "John Doe")` → Stores a session attribute.
5. `application.getServerInfo()` → Retrieves server details.

JSP Code	Explanation
`<%@ page language="java" contentType="text/html;`	This directive tells the JSP engine that we are using Java. It also sets the response content type to HTML
`<!DOCTYPE html>`	Declares the document type as HTML.
`<html><head><title>JSP Implicit Objects Example</title></head>`	Defines the HTML structure and sets the page title.
`<body>`	Starts the HTML body.
`<h2>Using JSP Implicit Objects</h2>`	Displays a heading on the webpage.

Using `request` Object

JSP Code	Explanation
`<p><b>Client IP Address:</b> <%= request.getRemoteAddr() %></p>`	`request.getRemoteAddr()` fetches the client's IP address and displays it on the webpage.
Using `response` Object	
`<% response.setContentType("text/html"); %>`	Ensures that the response is returned in HTML format. (Though not necessary here, it's good practice.)
Using `out` Object	
`<% out.println("<p><b>Using out object:</b> Hello, this is a JSP implicit object example!</p>"); %>`	Prints text using `out` (which is a JSP implicit object for output).
Using `session` Object	
`<% session.setAttribute("user", "John Doe"); %>`	Stores `"John Doe"` in the session under the key `"user"`.
`<p><b>Session Attribute (user):</b> <%= session.getAttribute("user") %></p>`	Retrieves and displays the `"user"` session attribute.
Using `application` Object	
`<p><b>Server Info:</b> <%= application.getServerInfo() %></p>`	Fetches and displays the server's name and version using `application.getServerInfo()`.
`</body></html>`	Ends the HTML document.

Key Implicit Objects Used

1. **`request`** → Retrieves client details like IP address.
2. **`response`** → Sets HTTP response parameters.
3. **`out`** → Sends output to the client.
4. **`session`** → Stores user-specific data across multiple requests.
5. **`application`** → Retrieves application-wide settings and server information.

Directive elements in jsp

These directives provide directions and instructions to the container, telling it how to handle certain aspects of the JSP processing.

A JSP directive affects the overall structure of the servlet class. It usually has the following form −

<%@ directive attribute = "value" %>

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.

2 **<%@ include ... %>**
Includes a file during the translation phase.

3 **<%@ taglib ... %>**
Declares a tag library, containing custom actions, used in the page

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

1. <%@ page attribute="value" %>

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class,interface or all the members of a packag
similar to import keyword in java class or interface.

Example of import attribute

```
1. <html>
2. <body>
3.
4. <%@ page import="java.util.Date" %>
5. Today is: <%= new Date() %>
6.
7. </body>
8. </html>
```

errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

```
1. //index.jsp
2. <html>
3. <body>
4.
5. <%@ page errorPage="myerrorpage.jsp" %>
6.
7.  <%= 100/0 %>
8.
9. </body>
10. </html>
```

---

isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
1. //myerrorpage.jsp
2. <html>
3. <body>
4.
5. <%@ page isErrorPage="true" %>
6.
7.  Sorry an exception occured!<br/>
8. The exception is: <%= exception %>
9.
10. </body>
11. </html>
```

Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive

1. <%@ include file="resourceName" %>

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
1. <html>
2. <body>
3.
4. <%@ include file="header.html" %>
5.
6. Today is: <%= java.util.Calendar.getInstance().getTime() %>
7.
8. </body>
9. </html>
```

The `taglib` directive

in JSP is used to import and use **custom tags** or **JSP tag libraries** (like JSTL – JavaServer Pages Standard Tag Library).

```
<%@ taglib uri="URI_of_library" prefix="prefix_name" %>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>

<html>

<head>

  <title>JSP taglib Example</title>

</head>

<body>
```

```
<h2>Using JSTL with taglib</h2>

<%-- Declare a variable using JSTL --%>
<c:set var="message" value="Hello, JSP Tag Library!" />

<%-- Output the variable using c:out --%>
<p>Message: <c:out value="${message}" /></p>

</body>
</html>
```

Output

Using JSTL with taglib

Message: Hello, JSP Tag Library!

Exception Handling in JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive

Example of exception handling in jsp by the elements of page directive

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for handling the exception

index.jsp

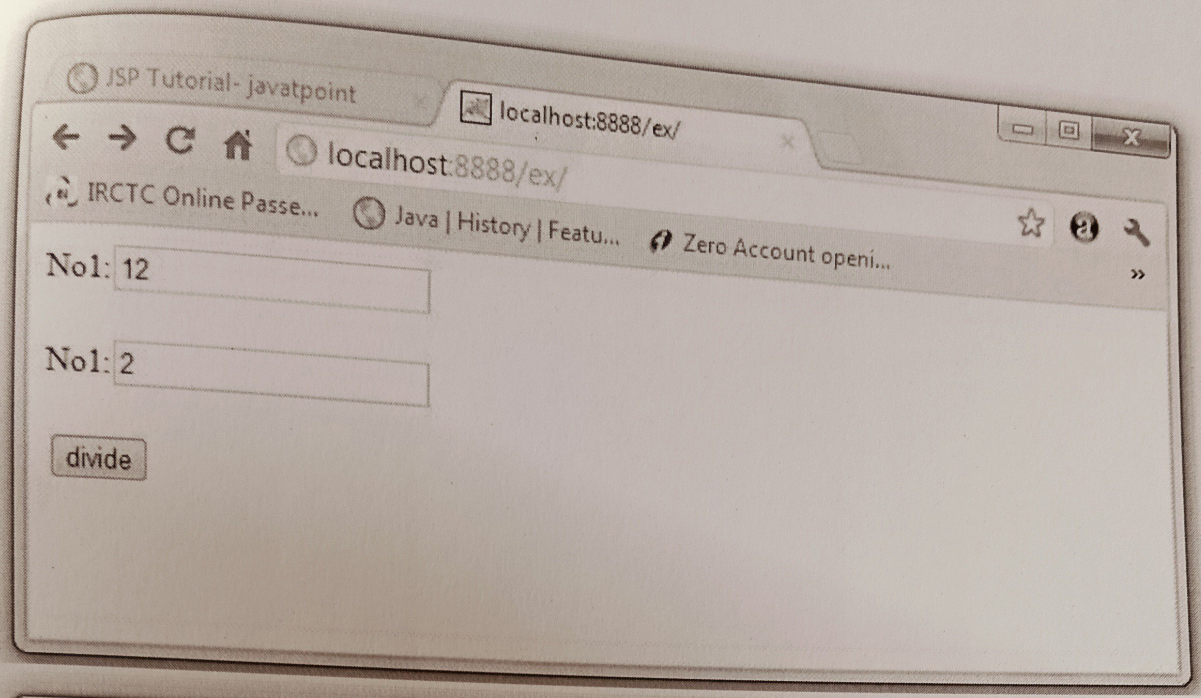
```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

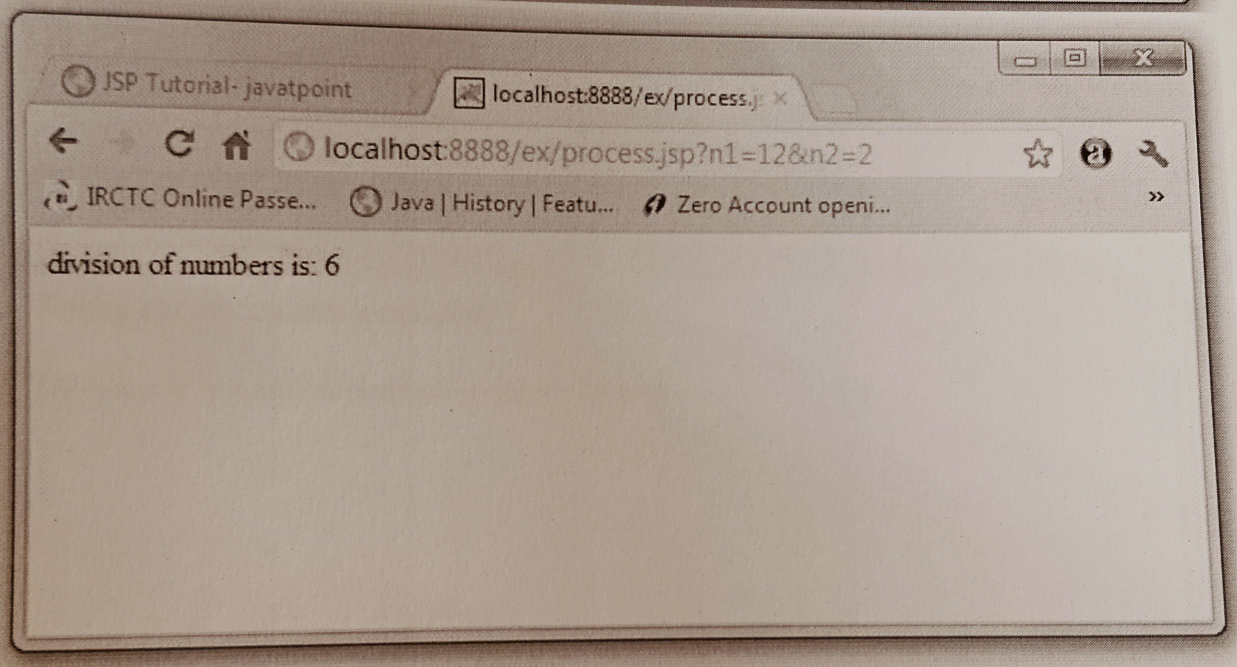
process.jsp

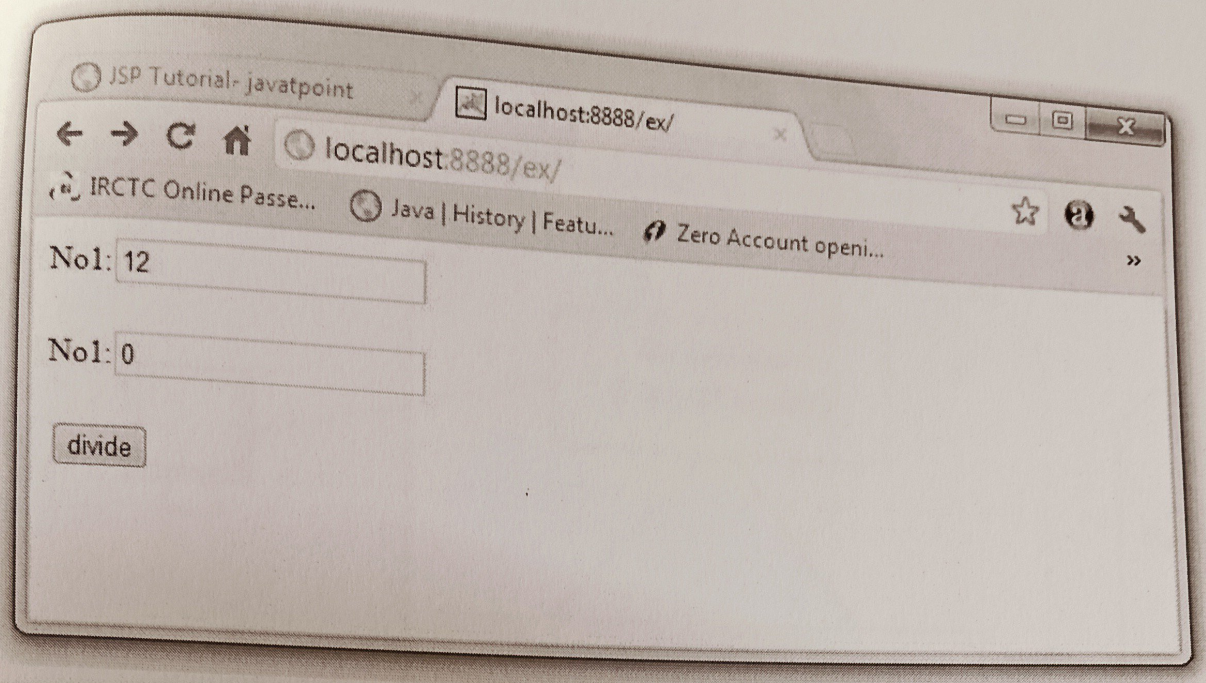
```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

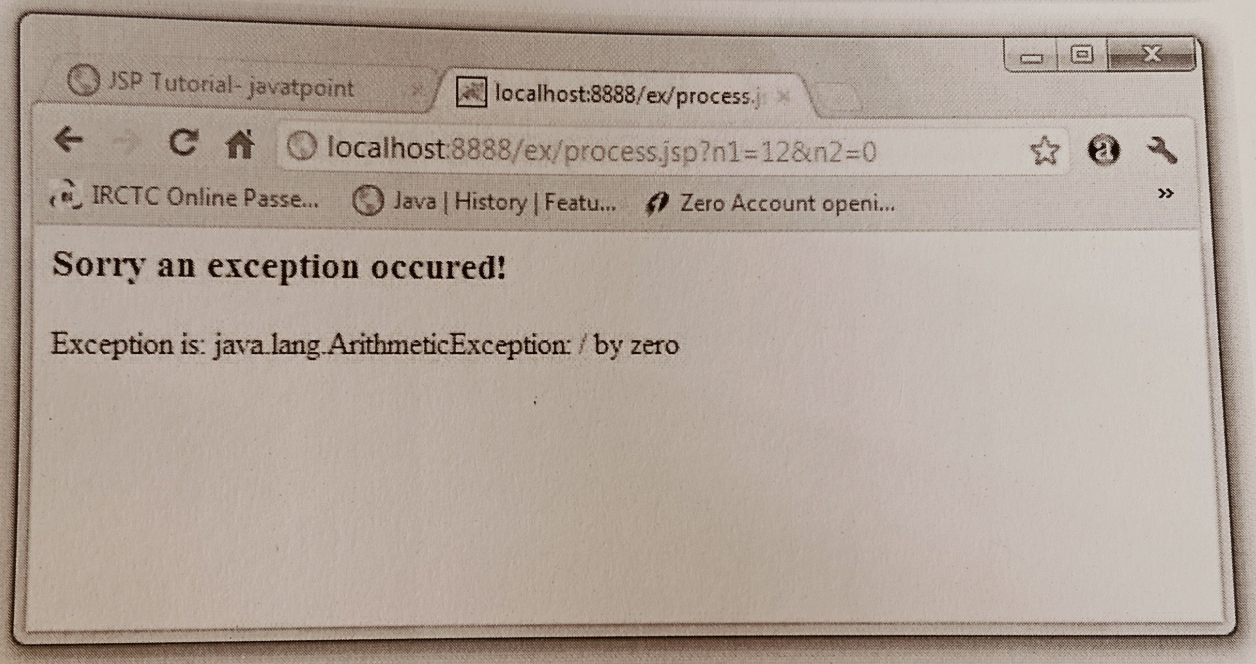
error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```


JSP Tutorial- javatpoint
localhost:8888/ex/
localhost:8888/ex/
IRCTC Online Passe...
Java | History | Featu...
Zero Account openi...
No1: 12
No1: 2
divide


JSP Tutorial- javatpoint
localhost:8888/ex/process.j
localhost:8888/ex/process.jsp?n1=12&n2=2
IRCTC Online Passe...
Java | History | Featu...
Zero Account openi...
division of numbers is: 6





MVC in JSP

MVC stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.

Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model represents the state of the application i.e. data. It can also have business logic.

View represents the presentaion i.e. UI(User Interface).

Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application