Servlets and Pages

Presentation Layer

Content

- Java Servlets
- Java Server Pages (JSP)
- JSP Standard Tag Library (JSTL)
- Expression Language

Java Servlets

What is Servlet?

- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

- Class that implements Servlet interface

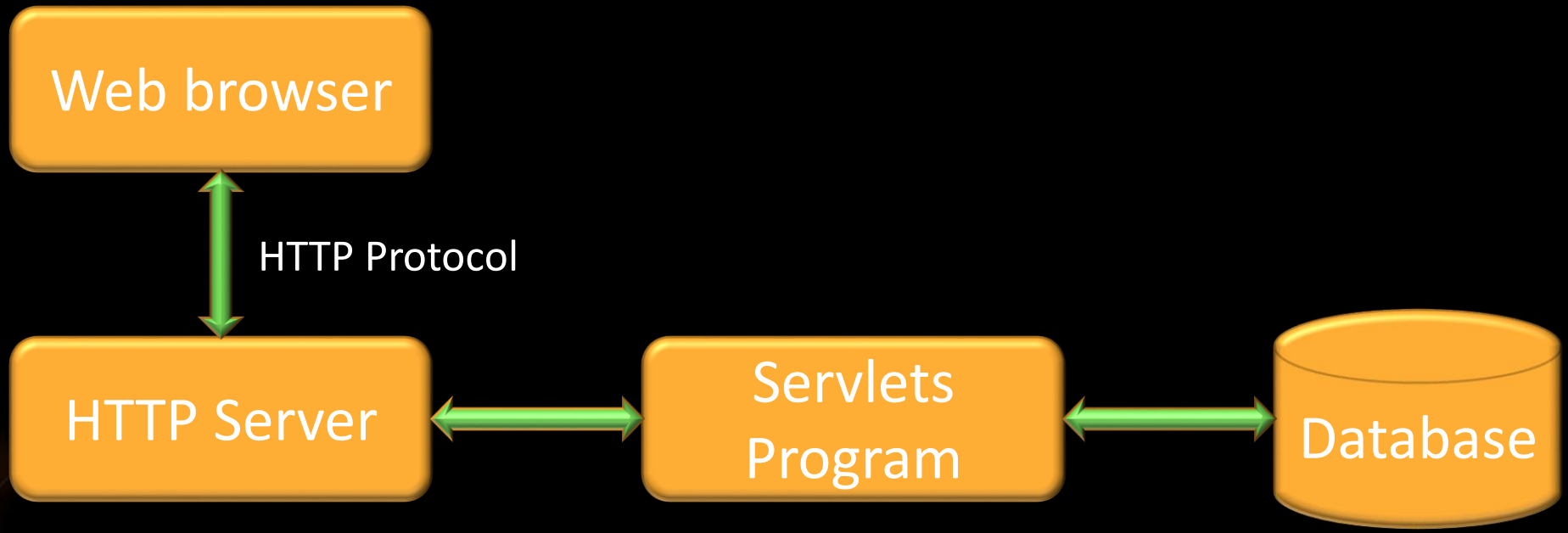
- Can be used in all kinds of requests (GenericServlet), but most widely used together with the HTTP protocol (HttpServlet)

Servlet applications

- Generating dynamic content
- Processing HTTP requests
- Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.
  - Performance is significantly better.
  - Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
  - Servlets are platform-independent because they are written in Java.
  - Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
  - The full functionality of the Java class libraries is available to a servlet
- Base of the JSP and JSF technologies

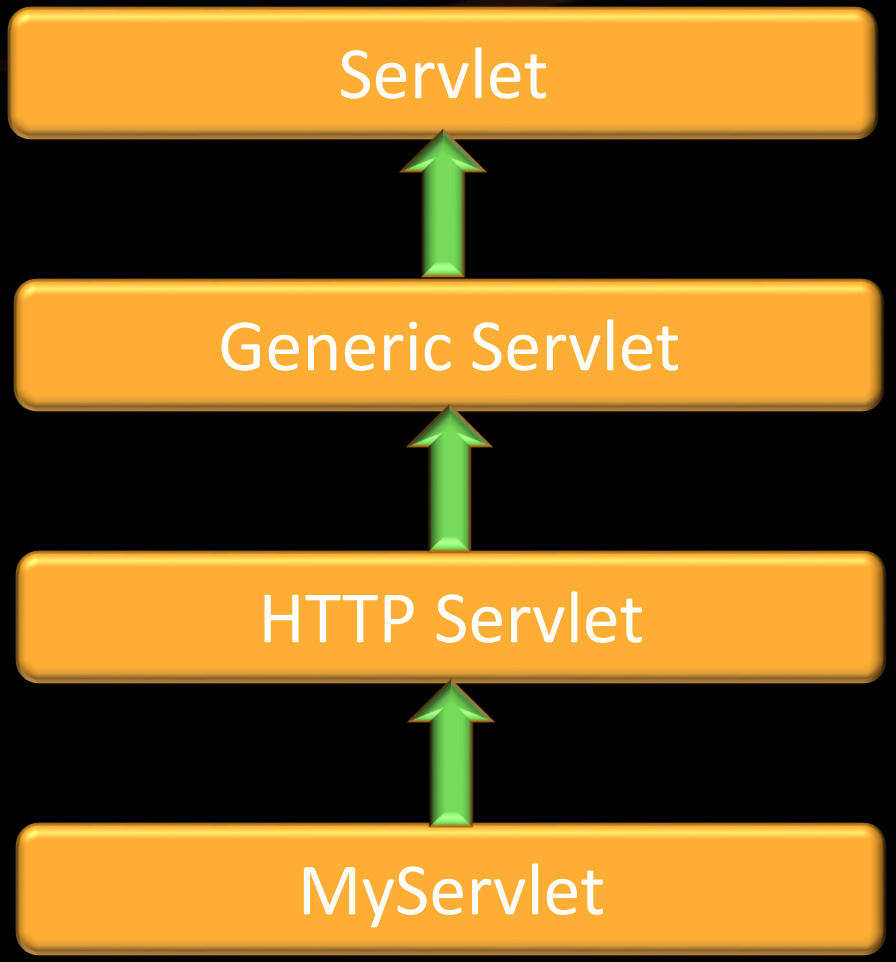
Servlets Architecture

- Following diagram shows the position of Servlets in a Web Application.



Servlet Hierarchy

- Servlet interface defines the base methods:
  - init(ServletConfig)
  - service(ServletRequest, ServletResponse)
  - destroy()
- GenericServlet implements the base functionality:
  - getInitParameter(String)
  - getServletContext()
- HttpServlet implements the HTTP protocol:
  - doGet(), doPost(), doPut(), doDelete()…
- MyServlet – user servlets most often inherit HttpServlet



Servlet's Life Cycle

- load – the container loads the servlet class during the start of the web module or during the first request
- instantiate – the container create instance of the servlet class using it's constructor
- init – before the servlet could service requests the container initialize it (using the init() method)

Servlet's Life Cycle (2)

- service – after successful initialization the servlet can process requests; the container creates new thread for every request and calls the service() method of the servlet

- destroy – when the servlet is no longer needed, the container calls the destroy() method

- unload – the servlet class is released from the JVM of the container

Data Scopes

- Web components data is kept as attributes to 4 scope objects
- Data is manipulated through setAttribute() and getAttribute() methods
- The scopes are (decreasing visibility):
  - application (javax.servlet.ServletContext) – data is accessible to all web components in the application
  - session (javax.servlet.http.HttpSession) – data is accessible to web components in the current session
  - request (javax.servlet.ServletRequest) – data is accessible to web components in the current request
  - page (javax.servlet.jsp.JspContext) – data is accessible only in the current JSP page

Servlet Concurrency

- The multithreading is part of the servlet specification
  - Only one instance is created from servlet class
  - Each request is processed in separate thread
  - All threads are working over the same object
- Concurrent access can happen in the following cases:
  - Access to attributes in the application scope
  - Access to attributes in the session scope
  - Access to variables in the servlet

Request and Response

- ServletRequest gives access to:
  - HTTP headers of the request
  - HTML form data and request parameters
  - Other client data (cookies, path, etc.)
- ServletResponse is responsible for the communication from the servlet back to the client:
  - Setting length и MIME (media) type
  - Sending data through the response (ServletOutputStream or Writer)

Java Servlets

Demo

Java Server Pages

Java Server Pages (JSP)

- Technology for creating static and dynamic views
- Uses HTML syntax and can be easily integrated with JavaScript and CSS
- Translates the content to servlet that is executed on the server and returns the generated representation
- Eases creation of views – less Java code is required
- Provides portability – most Java EE servers supports the JSP specification

The JSP idea

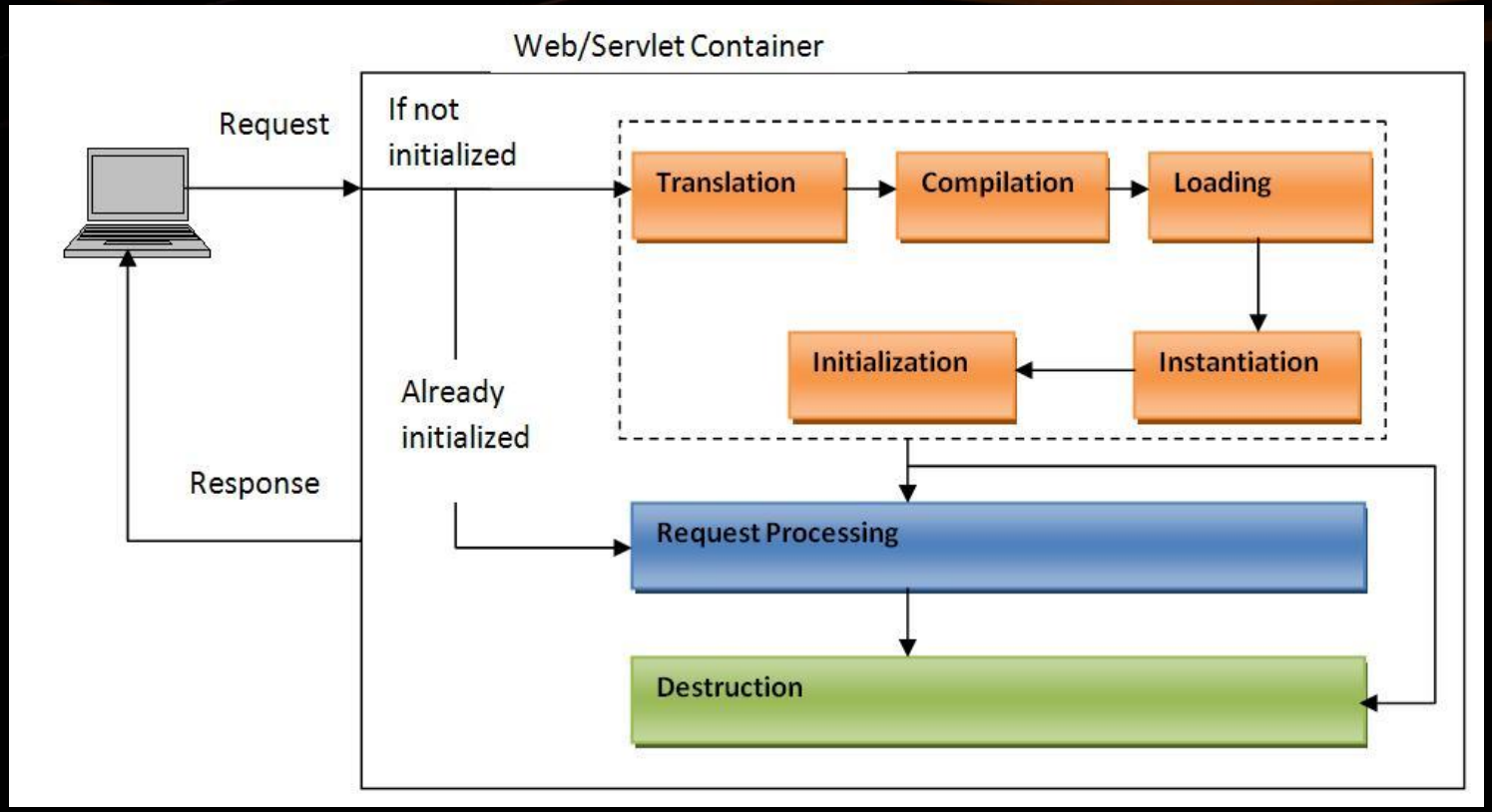
- JSP is based on the servlet technology
- Each JSP is in fact HTML page with special JSP tags that can have Java code
- The JSP file has .jsp extension (or .jspf for fragments)
- The JSP engine parses the JSP file and creates Java servlet. That servlet is later compiled to .class file

JSP Example

- Current date and time is shown with every request

```
<html>
   <head><title>Date JSP example</title></head>
   <body>
      The date is: <%= new java.util.Date() %>
   </body>
</html>
```

JSP Life Cycle



- jspInit(), jspDestroy()
- jspService(HttpServletRequest, HttpServletResponse)

JSP Default Variables

- The following variables can be used in the JSP by default:
  - request – current HttpServletRequest
  - response – current HttpServletResponse
  - session – current HttpSession (ако има такава)
  - out – text stream for JSP result (PrintWriter)
  - application – ServletContext of the application
  - exception – accessing possible errors
  - config, pageContext, page

JSP Directives

- JSP directives changes the structure of the generated servlet class
- Syntax: <%@directive attribute="value" %>
  <jsp:directive.name attribute="value"/>
- page directive – defines attributes for the JSP
  - imports packages: <%@ page import="java.util.*" %>
  - sets content type to the result: <%@ page contentType="text/plain" %>
  - sets the JSP as part of the HTTP session: <%@ page session="true" %>
  - sets JSP URL in case of errors: <%@ page errorPage="url" %>

JSP Directives(2)

- include directive – allows to include other pages
  - example: <%@ include file="/common/header.jsp" %>

- taglib директива – declares a tag library used in the page
  - example: <%@ taglib uri="uri" prefix="op" %>

- XML directive format:
  - <jsp:directive.page attribute="value" />
  - <jsp:directive.include file="url" />
  - <jsp:directive.taglib uri="uri" prefix="op"/>

JSP Actions

- XML elements that control the servlet behavior
- syntax: <jsp:name attribute="value" />

- examples:
  - <jsp:include page="relative URL" flush="true" />
  - <jsp:forward page="url" />
  - <jsp:useBean id="opa" />
  - <jsp:setProperty name="opa" property="prop" value="val" />
  - <jsp:getProperty name="opa" property="prop" />
  - <jsp:text>Template data</jsp:text>

JSP Declarations

- JSP declarations allow defining of methods or variables in the body of the generated servlet class
- syntax: <%! Java declarations; %>
- Usually does not generates content but used together with JSP expression and scriplets

- example:

```
<%!
   long counter = 0;
   public void getCounter() {return counter;}
%>
```

JSP Expressions

- JSP expressions are used to directly include the result of Java expression in the result of the JSP
- syntax: <%= Java expression %>

- examples:
  - <%= new java.util.Date() %>
  - <%= Math.PI %>
  - <%= request.getRemoteHost() %>
  - <%= session.getMaxInactiveInterval() %>

JSP Scriplets

- JSP scriplets allows insert of Java code in the _jspService() method of the JSP
- syntax: <% Java code; %>
- Scriplets have access to the standard JSP variables (request, response, session, …)
- example:

```
<% for (int i=0; i<10; i++) { %>
   <%= i %> * <%= i %> = <%= i*i %>
   <br>
<% } %>
```

Java Server Pages

Demo

JSP Standard Tag Library

Custom tags

- Reusable components allowing component-oriented development in Java web applications
- Custom tag can be created by everyone
- Hides complexity of the visualization
- Looks like HTML tags which eases the web designers and developers

- example:
  <mytags:opa attribute="value" />

Tag library

- Collection of custom tags
- Each library has prefix and URI identifier
- Each tag library contains:
  - Tag library descriptor (TLD) – XML document describing the tags in the library
  - JAR containing the compiled classes and tag resources

- Usage:
  - TLD descriptor is placed in /WEB-INF
  - JAR is placed in /WEB-INF/lib
  - Library is registered in the JSP: `<%@ taglib prefix="mytags" uri="WEB-INF/my.tld" %>`
  - Tag is used: `<mytags:opa attribute="value" />`

JSP Standard Tag Library (JSTL)

- Contains reusable custom tags
- Standard tag library with JSP 2.0
- Implements often used functionalities:
  - JSTL Core – common functions for variables, conditions, cycles, etc.
  - JSTL Format – formatting and locales
  - JSTL XML – reading XML data, XML transformations
  - JSTL SQL – executions of SQL queries
  - JSTL Functions – execution of standard functions

JSTL Advantages

- Allows JSP pages to contain only XML
  - avoids placement of Java code in the views
  - the code is easier to read and support
- Eases the developers by giving them commonly used functionality out of the box
- Easy to use:

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

JSTL Core Tags

- Variables:
  - <c:set var="n1" value="v1" scope="page" />
  - <c:set var="n2" value="v2" scope="request" />
  - <c:set var="n3" value="v3" scope="session" />
  - <c:set var="n4" value="v4" scope="application"/>
  - <c:remove var="n1" scope="page" />
- Data visualization:
  - <c:out value="${user.age}" escapeXml="false"/>
- Error handling:
  - <c:catch var="DivideByZeroException">
    <% int x = 10/0; %>
    </c:catch>

JSTL Core Tags(2)

- Conditional operations:
  - <c:if test='${param.p == "val"}'>
    </c:if>
  - <c:choose>
      <c:when test='${param.p == "val"}'> </c:when>
      <c:otherwise> </c:otherwise>
    </c:choose>
- Cycles:
  - <c:forEach var='item' begin='1' end='10'>
      <c:out value='${item}'/>
    </c:forEach>
  - <c:forEach var='item' items='${itemsList}'>
      <c:out value='${item}'/>
    </c:forEach>

Tag files

- File containing reusable JSP fragment
- Standard extension is .tag
- Usage:
  - dividing the JSP into modules
  - reusing page logic
- Types:
  - without body: <m:opa />
  - with body: <m:opa>X</m:opa>

Tag Files Example

- File with name opa.tag:

```
<%@ tag body-content="empty" %>
<%@ attribute name="firstName" required="true"%>
<h1>Hello, <%= firstName %> !</h1>
```

- Placement of opa.tag in /WEB-INF/tags
- Declaration:

```
<%@ taglib prefix="m" tagdir="/WEB-INF/tags" %>
```

- Usage:

```
<m:opa firstName="Penn"/>
```

JSP Standard Tag Library

Demo

Expression Language

Unified Expression Language (UEL)

- Allows access to objects, their properties and method from JSP pages
- Syntax: ${ el израз }
- Replaces JSP action tags:
  - <jsp:useBean id="user" scope="request" />
  - <jsp:setProperty name="user" property="name" value="Penn" />
  - <jsp:getProperty name="user" property="name"/>
- Example:
  - ${opa.prop = "Penn"}
  - ${opa.prop}

UEL Advantages

- Short access notation to access data in the scopes (page, request, session and application)

  ${course}

- Easy access to collection elements

  ${students[2]}

- Easy access to bean object properties:

  ${course.presenter.name}

- Easy access to request parameters (headers, cookies, etc.)

  ${param["studentCount"]}

  ${cookie["dateOfLastVisit"]}

UEL Advantages(2)

- Simple operators (+, -, *, /, =, <, >, ==, &&, ||, ?:, empty, not):
  ${(2 + 5) * 3}
  ${course.studentCount - 1}
  ${not empty presenter ? "here" : "late" }
- Automatically converts to string
- Calling functions ability:
  ${fn:length("It's the cycle of life…")}
- Returns empty string instead of exception:
  ${thisIsNull.prop} → ""

UEL Object Access

- Attributes can be looked up in specific scope:
  ${session.course}
- If attribute name is not valid Java identifier, alternative notation is used:
  ${session[course.with.dots]}
- If scope is not given the look up is done from the lowest to highest scope:
  - pageContext
  - request
  - session
  - application

Expression Language

Demo