Java EE

* Java EE is an open, standards-based development and deployment platform for creating distributed, transactional, reliable, secure, multi-tiered, web-based, server-centric, component-based enterprise applications
* Java EE Application Model
* Java programming language, Java Virtual Machine (JVM)
* Java EE Components
* Java EE Clients
* Application Clients, Applet (embedded in web clients)
* Web Components
* Servlets, JavaServer Pages (JSP), JavaServer Faces (JSF)
* Enterprise JavaBeans (EJBM)
* Java EE Containers
* Client containers, EJB container
* Java EE Server
* Java Web Application
* Collection of resources installed under a specific subset of the URL namespace of a web application server complaint with the java EE Specification (e.g. Apache’s Tomcat, Apache’s Geronimo, Sun Microsystems’ Glassfish =, IBM’s WebSphere, etc.)
* Resources
* Static resources: web poages, images, stylesheets, etc (serves as is)
* Dynamic resources: servlets, JSPs
* Miscellaneous resources: business object classes (e.g. Java Beans, EJB), support libraries etc
* XML-formatted descriptor and configuration files
* Web.xml, application.xml, context.xml, etc
* Organized into a standard hierarchical structure and typically packaged and deployed as

WAR or EAR files

* Java EE APIs
* Enterprise JavaBeans Technology
* Java Servlet Technology
* Java Server Pages
* Java Server Pages Standard Tag Library
* Java Server Faces
* Java Msg Service API
* Java Transaction API
* JavaMAil API
* JavaBeans Activation Framework
* Java API for XML Processing

Servlets

* Java Object based on the Servlet API
* Runs in a server application to answer client request; technically, servlets are not tied to a specific-client-server protocol, but they are most commonly used with HTTP and the term ‘servlet’ is often used in the context of an “HTTP Servlet”
* Web-tier components in the Java EE architecture
* Runs in, and is managed by a web-tier container called the ‘Servlet Container’
* Mapped to URLs to which clients send requests
* Typically asked with (among other things)
* Processing and/or storing data submitted vial HTML forms
* Generating dynamic content
* Javax.servlet
* Servlet, GenericServlet
* ServletRequest, ServletResponse
* ServletConfig, ServletContext
* RequestDispatcher
* Javax.servlet.http
* HttpServlet
* HttpServletRequest
* HttpServletResponse
* HttpSession
* Cookie
* Servlet Processing
* Client sends a request to a web server URL that is mapped to a servlet. Web server passes on the request to the servlet container
* Servlet container checks if servlet is already loaded
* If it is not yet loaded, servlet container loads the servlet class and instantiates the servlet, and calls its **init** method.
* Servlet container invokes the servlet’s service method, passing request and response objects as arguments
* Servlets processes the request using the response object to create the response, which is returned by the servlet container to the web server, which in turn sends the response to the client
* Subsequent request to the servlet will not require servlet re-instantiation, unless the servlet has been unloaded; before a servlet is web server, which in turn sends the response to the client
* init (config)
* Invoked once on the servlet by the servlet container when the servlet is instantiated; can be used by the servlet for one-time startup initialization
* service (request, response)
* Invoked each time the servlet is called upon to process a request (typically on a separate thread for each call)
* In HttpServlet, the default Service implementation maps the call to a specific doXXX() method (e.g. doGet, doPost) which is typically overridden to affect the servlet’s functionality
* Destroy()
* Invoked on the servlet by the servlet container when the servlet is to be unloaded (e.g. when the application is stopped or undeployed); can be used by the servlet for clean-up processing (e.g. resource deallocation)
* Servlet Request Processing (HttpServletRequest)
* Retrieving user-supplied request parameters
* Retrieving request header values
* ServletResponse Processing (HttpServletResponse)
* Setting response status code
* Setting response headers
* Obtaining output object for sending the response
* Servlet Request Dispatching (requestDispatcher)
* Obtain a RequestDispatcher to a resource (static or dynamic) from the request object

RequestDispatcher rqstDsp;

rqstDsp = request.getRequestDispatcher(res);

* Include the dispatcher resource (or its output) in the current response; one or more resources can be included (e.g. use for banners, footers, etc)

rqstDsp.include(request, response);

* Forwards the processing of the current request to the dispatcher resource; the servlet processing the current request must not generate a response (e.g. use in MVC “controller” servlets)

rqstDsp.forward(request, response);

* Session Tracking (HttpSession)
* Session tracking support is implemented either cookies or URL-rewriting
* Obtaining session object from the current request

HttpSession session;

session = request.getSession(createNew);

* Obtaining session information (HttpSession)
* getCreationTime(), getLastAccessedTime(), getMaxInactiveInternal(), getId(), isNew(), setMaxInactiveInterval(int val)
* destroying sessions
* invalidate()
* URL-rewriting (HttpServletResponse)
* encodeURL(String url), encodeRedirectURL(String url)

Web Context (ServletContext)

* a web application is assiociated with a context, which is an object that provides methods that servlets use yo communicate with the servlet container
* obtaining the servlet context(HttpServlet)

ServletContext context;

context = this.getServletContext();

* obtaining context information(ServletContext)
* getServerInfo(), getContextPath(), getRealPath(), getResource(), getResourceAsAtream(), getMimeType, getInitParameter(), getInitParameterNames(), getRequestDispatcher(), getContext()

Servlet configuration (servletConfig)

* getServletName(), getServletContext(), getInitPArameter(), getInitParameterNames()

Information sharing using scope objects

* A request may be processed by a several web application components (e.g. through calls to RequestDispatcher forwad/include) and there may be a need for one component to communicate information to the other components in the request processing chain.
* A client session typically consists of multiple request, which due to the stateless nature of HTTP, will appear to the application as being “unrelated” to one another; the HttpSession object can be used to “relate” these requests together, but there may still be a need to share information created in one request with a subsequent request
* Different web application components may require access to common resources or information (e.g. page counters, shared database connection)
* Information sharing is accomplished by creating attribute objects and exposing these objects in the appropriate scope.
* Scopes:
* Request scope(HttpServletRequest)
* Session scope (HttpSession)
* Web Applicationor Web Context scope (ServletContext)
* Page scope (local objects in the servlet)
* Creating, accessing, and removing attribute objects
* setAttribute (String attrName, Object attrValue)
* getAtrribute (string attrName)
* getAttributeNames()
* removeAttribute (String attrName()

Advanced Servlet Topics

Listeners

* java objects used to “subscribe” to application “events” in order to be “notified” when these events occur
* context-related events
* context initialized, context destroyed, context attribute changes
* session-related events
* session created, session destroyed, session attribute changes
* request-related events
* request initialized, request destroyed, request attribute changes

javax.servlet

* ServletContextListener, ServletContextAttributelistener
* ServletRequestListener, ServletRequestAttributeListener

javax.servlet.http

* HttpSessionListener, HttpSessionAttributeListener

Filters

* Java objects used o intercept incoming requests and outgoing responses in order to perfeorm various tasks such as:
* Authentication and access control
* Logging, auditing
* Caching, data compression
* Content Transformation
* Filter objects are mapped to the URL patterns they are intended to intercept
* Filter objects can be “chained” together

javax.servlet

* Filter, FilterChain, FilterConfig

Java Server Pages

* Simply an HTML web page that contain additional bits of code that execute application logic to generate dynamic content.
* Java Server Pages Actions (JSP tags) perform a variety of functions and extend the capabilities of JSP.
* Java Server Pages Actions use XML-like syntax, and are used to manage JavaBeans component.
* Directives are instructions that are processed by the JSP engine when the page is compiled to a servlet.
* Directives are used to set page-level instructions, insert data from external files and specify custom tag libraries
* <%@ %>
* Motivation
* It is typically a good idea to separate business logic from presentation cconcern
* Allows modern web development teams to be divided up into programmers and web page authors / designers
* Fosters component reuse (e.g. the same data object can be consumed by user agents of varying capabilities and needs)
* Servlets can be very powerful for programming business logic, but are very awkward to use when generating static (i.e. template) content.
* (X)HTML marked-up documents are very convenient for static content generation but cannot be used to program business logic (or generate dynamic content arising from data produced by the business logic).
* Features
* Text-based document capable of generating both static and dynamic content (typically intermixed)
* Mark-up based document syntax (JSP-style or XML-style), combining (X) HTML elements as well as standard and custom JSP elements; thus, web page authors can feel right “at home” with the mark-up syntax.
* Embedded Java Coding support via “scriptlets”
* <% %>
* Template text are converted into JSPWriter
* Components
* Template (i.e. static) text
* JSP elements
* Directives
* <%@ page ContentType=”text/html” pageEncoding=”UTF-8” %>
* <%@ page import=”java.util.Random” %>
* autoFlush
* buffer
* contentType
* errorPage
* extends
* import
* info
* isELIgnored
* isErrorPage
* isThreadSafe
* language
* pageEncoding
* session
* <%@ taglib uri=”<http://java.sun.com/jsp/jstl/core>” prefix=”c” %>
* prefix
* taglib
* uri
* Scripting Elements
* Declarations
* <%! int a = 100; %>
* <%! int square(int n) (return n\*n ; ) %>
* Expressions
* <% String s = new java.util.Date() . toString(); %>
* Scriptlets
* <% for(int i = 0; i < 10; i++) {out.println(i); } %>
* Actions
* Standard actions
* <jsp:directive.include>, <jsp:directive.page>
* <jsp:declarations>
* <jsp:scriplet>
* <jsp:expression>
* <jsp:include>, <jsp:forward>
* <jsp:useBean>, <jsp:setProperty>, <jsp:getProperty>
* <jsp:plugin>, <jsp:param>, <jsp:params>, <jsp:fallback>
* <jsp:element>, <jsp:attribute>, <jsp:body>
* <jsp:text>
* Custom Actions (JSTL)
* JSTL, user-written custom tag libraries
* Expression Language (EL)
* ${ }
* Implicit Scripting Objects
* request, response, out, pageContext
* session, pageContext, application
* config, page, exception
* Implicit EL Objects
* pageContext
* pageScope
* requestScope
* sessionScope
* applicationScope
* param, paramValues
* header, headervalues
* cookie
* initparam
* Comments
* <% - - this is a JSP comment - -%>

Servlets or JSPs?

* The common practice is to leverage both technologies to implement the MVC design pattern

Model-View-Controller (MVC) Design Pattern

* Model
* Represents business objects (logic and state)
* View
* Presentation of the model in some appropriate way
* Controller
* Mediates application flow

A sample web MVC framework can be use

* JavaBeans for the model
* JSPs for the View
* Servlets for the controller

JSP Standard Tag Library (JSTL)

* Set of custom JSP elements that provide various programmatic functionality via mark up syntax
* Core Tag Library
* Variable support, flow control, URL management
* SQL Tag Library
* Database connections, queries, updates
* Internationalization Tag Library
* Locate setting, message bundling, number formatting, date formatting
* XML
* Core XML processing, flow, control, transformation
* JSTL Function
* String functions, collection length
* In addition to JSTL, developers can also create their own tag libraries for commonly occurring tasks

ACRONYMS:

Adobe AIR (Adobe Integrated Runtime)

AJAX (Asynchronous JavaScript and XML)

ANSI (American National Standards Institute)

ASP (Active Server Pages)

BOM (Browser Object Model)

CFML (ColdFusion Markup Language)

CGI (Common Gateway Interface)

DHTML (Dynamic HTML)

DOM (Document Object Model)

DTD (Document Type Definition)

EAR (Enterprise Archive)

EIS (Enterprise Information Systems)

EJB (Enterprise JavaBeans)

EL (Expression Language)

IIS (Internet Information Services)

JAR (Java Archive)

JSF (Java Server Faces)

JSON (JavaScript Object Notation)

JSP (Java Server Pages)

JSTL (JSP Standard Tag Libraries)

PERL (Practical Extraction and Reporting Language)

PHP (Hypertext Preprocessor)

RDF (Resource Description Framework)

RIA (Rich Internet Application)

RSS (Really Simple Syndication)

SMIL (Synchronized Multimedia Integration Language)

SMX (Server Macro Expansion)

SOAP (Simple Object Access Protocol)

SVG (Scalable Vector Graphics)

WAP (Wireless Application Protocol)

WAR (Web Application Archive)

WSDL (Web Services Description Language)

XForms (XML Forms)

XLink (XML Linking Language)

XPointer (XML Pointer Language)

XQuery (XML Query Language)

XSD (XML Schema)

XSL (Extensible Style Sheet Language)

XSL-FO (Extensible Style Sheet Language Formatting Objects)

XSLT (XSL Transformations)

XSRF (Cross-site request forgery)

XSS (Cross-site scripting)