What is J2EE?

- A set of standard Java APIs and reference implementations bundled as Java class libraries for enterprise computing
- Based on the Java 2 Standard Edition
- A software platform supporting multi-tiered architecture
  - Presentation Tier
  - Web Tier
  - Business Tier
  - Enterprise Information System Tier (e.g. database)
Overview of Java Standards

- J2SE – Containing JVM, tools and class libraries for generic Java application development
- J2EE – Containing tools and class libraries for Enterprise Java application development
- J2ME – Containing specialized JVM/KVM/JavaCard VM, tools and libraries for embedded and mobile devices
- API Standardization via Sun’s Java Community Process (JCP)

What are Enterprise Applications?

- A set of computing applications to perform business functions in an enterprise (accounting, sales, ERP, CRM, etc.)
- Enterprise computing based on a software platform beyond the basic client/server computing (naming, security, transaction, persistency, messaging, etc.)
- J2EE, a Java-based, standard platform for enterprise application integration (EIA)
  - Based on Software Component technologies – SWE 645 (Component/Container design pattern defined at each of application tier)
- Why a software platform is needed and how to create one?
Sample Server Serving Multiple Clients

Network Programming Basics

1. Client sends request to the server which expects incoming requests
2. Server spawns a “worker” thread and binds the client to a new socket
3. The worker thread takes charge of communicating with the client

How to reuse the server implementation for various apps?

Abstract Worker Functions

class WorkerThread extends Thread {
   // instance parameters …
   // constructors …
   public void run() { // execute worker function
      AppComponent component = new AppComponent(params);
      component.execute();
      .... }
}

- Server functions are “componentized” into various Java classes
- Application development becomes customizing or subclassing the standard component classes
Building an Application Server

Enterprise Software Platform

- What is a software platform
  - Reference implementation of a standard software framework (a set of standard APIs and protocols/behaviors)
  - Component-based computing environment for plug-and-play of application components

- Enterprise specific requirement
  - Client/Server computing in a network environment
  - Multi-user, multi-threaded service implementation
  - Standard utility (plumbing) functions for centralized control and governance (security, naming/directory, persistency, transactions, etc.)

- J2EE as a software platform
  - J2EE Application Server as a reference implementation of the Servlet and EJB specs, along with other API standards
Java Servlet APIs

- APIs to define the J2EE web tier
- Modeled after the CGI APIs
- Java Servlet Object – An web component to be driven by a worker thread of Web Container
- Important Java Servlet APIs
  - `javax.servlet.Servlet`
  - `javax.servlet.GenericServlet`
  - `javax.servlet.ServletConfig`
  - `javax.servlet.ServletContext`
  - `javax.servlet.http.HttpServlet`
  - `javax.servlet.http.HttpServletRequest`
  - `javax.servlet.http.HttpServletResponse`
  - `javax.servlet.http.HttpSession`
  - `javax.servlet.ServletException`

Development using Servlet APIs
- Extending the `HttpServletRequest` class
- Controlling the `HttpSession` to enable statefulness of the app

The HttpServlet Methods

protected void doGet(`HttpServletRequest` req, `HttpServletResponse` resp) throws `ServletException`, `IOException`
- Callback function to handle GET request message
protected void doPost(`HttpServletRequest` req, `HttpServletResponse` resp) throws `ServletException`, `IOException`
- Callback function to handle POST request message
protected void doPut...
protected void doHead...
protected void doOption...

public void init(`ServletConfig` config) throws `ServletException`
- Initialization of Servlet parameters
public void destroy()
- Remove the servlet from service (end of life-cycle)
### Java Servlet Example

```java
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        // getting request parameters
        String name = request.getParameter("name");
        // outputting response message
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.print("<html>
          <head>
          <title>Hello Servlet Example</title>
          </head>
          <body>
          Hello " + name + "
          </body>
          </html>");
    }
}
```

You need to have the `servlet-api.jar` file in your class path when compiling. It is included in the server environment by default.

### Corresponding input field (of the HTML form) with the name “name”

### Deployment Descriptor

- **What is it and why is it needed?**
  - XML configuration file for initializing servlet classes and defining servlet invocation URLs
  - Used for dynamic class loading
  - A J2EE standard with the name `web.xml` in the WEB-INF directory
- A deployment descriptor defines the followings
  - Servlet name and class bindings
  - Servlet name to URL mapping
  - Initialization of servlet parameters
  - Default error and welcome page
  - Security (user authentication and authorization roles)
Development Process

1. Compile the servlet
   - javac –classpath servlet.jar HelloServlet.java

2. Create the deployment descriptor file (web.xml)

3. Package web application files (html, jsp, servlet class, web.xml) into a war file (web application archive, same as jar format). A web app is defined by a single war file
   - jar -cvf helloworld.war my-webdev-directory
   - Web application directory – A J2EE standard
     - META-INF (directory for specifying metadata info)
     - Static web page files (html, jsp, images, etc.)
     - WEB-INF
       - classes (directory for servlet classes)
       - web.xml

4. Deploy to application server-specific directory

Java Server Pages (JSP)

- What is JSP and why is it useful?
  - Embedded Java code (specified with special tags) in HTML pages
  - When invoked, JSPs (with the extension .jsp) are converted into Java Servlet classes by J2EE server before returning the page content to client
  - Same concept as SSI (server-side include)
  - JSPs are primarily used in presentation layer. Useful in supporting MVC design pattern
Basic JSP Elements

- Page directives (<%@page ... %>) , configuring the page settings, such as content type
- Tag library directives (<%@taglib ... %>), importing tag libraries
- JavaBeans declaration (jsp:useBean), creating or referencing Java objects used in the page or passed from another servlet or page
- JSP expressions (${expression}), retrieving the value of parameters or object properties
- jsp:setProperty, setting the value of an object property
- Java statements (<%...%>), regular java code

Servlet Chaining

- Servlet can be chained to fine-tune application workflow control and create dedicated functional components (e.g. MVC and other control patterns)
- HttpServletRequest/Response objects are passed along the chain. Servlet can add/remove/modify attribute values using get/setAttribute(String name)
- Use RequestDispatcher API to enable chaining
  - RequestDispatcher.forward(HttpServletRequest req, HttpServletResponse res)
Java Server Faces (JSF)

- A server-side GUI component framework for Java technology-based web applications, providing a fine-tuned GUI rendering mechanism on top of existing Web component (Servlet, JSP)

- Basic JSF Elements:
  - A set of APIs for defining and referencing UI components, managing the UI component state, and handling user events, etc.
  - Two set of JavaServer Pages (JSP) custom tag libraries
    - h - JavaServer Faces component tags for all UICOMPONENT and HTML RenderKit Renderer combinations defined in the JavaServer Faces Specification
    - f - The core JavaServer Faces custom actions that are independent of any particular RenderKit for expressing UI components within a JSP page and for wiring components to server-side objects

Online References

- Designing Enterprise Applications with the J2EE Platform, 2nd edition
  - http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/

- Sun’s J2EE Tutorial

- J2EE specification
  - http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html

- Servlet and JSP specifications
  - JSPs http://jcp.org/en/jsr/detail?id=245
Summary

- J2EE as a software platform provides a standard way for enterprise application development and integration.
- J2EE is based on a multi-tier architecture, with its web tier standardized in Web container and components (servlet and JSPs).
- Web applications are packaged in a war file with standard directory structure. Each app is configured via the deployment descriptor in standard XML format.
- Servlet APIs are modeled after the HTTP. The request and response messages are encapsulated as HttpServletRequest and HttpServletResponse objects.
- JSPs are converted into servlet classes first before being returned to the requesting client.
- Servlet/JSPs can be chained to enable fine-tuned modularization and object reuse.

Quiz

- Are Servlet methods thread-safe? Why?
- What is a J2EE web application and what are the steps to create one?
- What is the API for retrieving input parameter values from an HTML form in a servlet?
- How to create servlet chaining and why is it useful?
- Write a simple servlet application that returns the total number of user hits of the servlet.