Enterprise Computing with Java

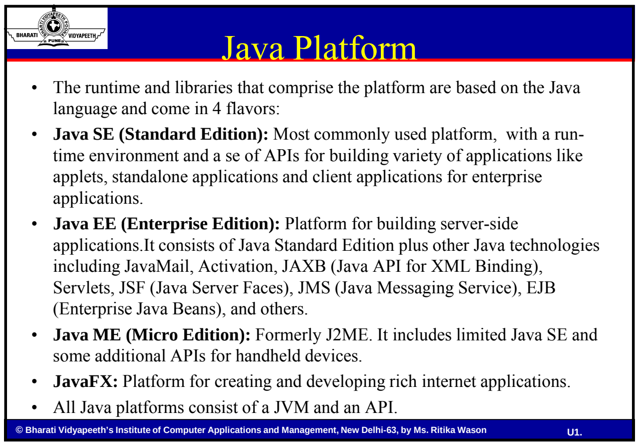
MCA-305
UNIT I

Learning Objectives

- Introduction to J2EE
- MVC architecture
- Servlets and it's life cycle
- Problems with cgi-perl interface
- Generic and http servlet
- Servlet configuration
- Various session tracking techniques
- Servlet context
- Servlet configuration
- Servlet colloboration.

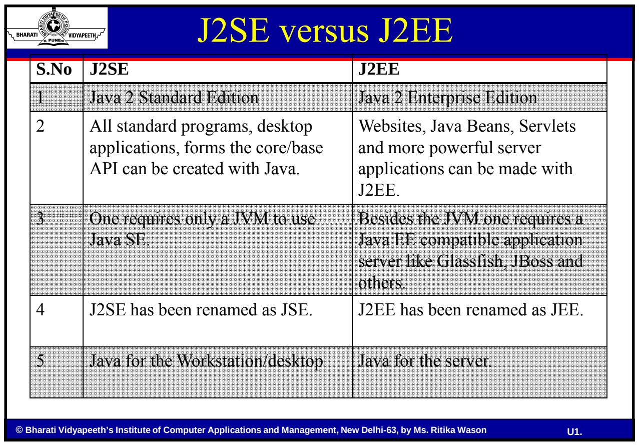
Java, J2SE, J2EE and J2ME

- "Java" refers to both a language and a platform.
- The Java programming language is a high-level, object-oriented language that has a particular syntax and style.
- A Java platform is a particular environment in which Java programming language applications run.



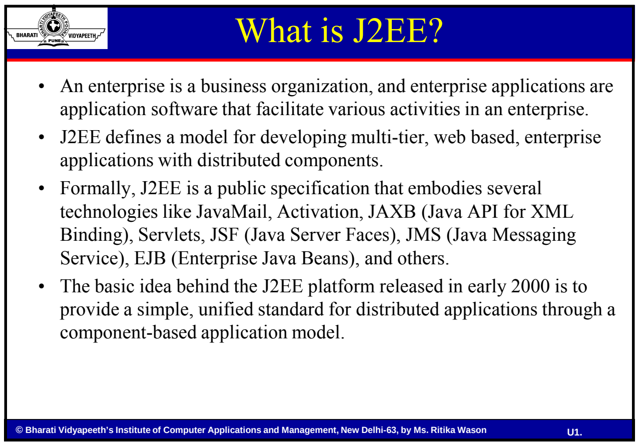
Java Platform

- The runtime and libraries that comprise the platform are based on the Java language and come in 4 flavors:
- **Java SE (Standard Edition):** Most commonly used platform, with a run-time environment and a se of APIs for building variety of applications like applets, standalone applications and client applications for enterprise applications.
- **Java EE (Enterprise Edition):** Platform for building server-side applications.It consists of Java Standard Edition plus other Java technologies including JavaMail, Activation, JAXB (Java API for XML Binding), Servlets, JSF (Java Server Faces), JMS (Java Messaging Service), EJB (Enterprise Java Beans), and others.
- **Java ME (Micro Edition):** Formerly J2ME. It includes limited Java SE and some additional APIs for handheld devices.
- **JavaFX:** Platform for creating and developing rich internet applications.
- All Java platforms consist of a JVM and an API.



J2SE versus J2EE

S.No	J2SE	J2EE
1	Java 2 Standard Edition	Java 2 Enterprise Edition
2	All standard programs, desktop applications, forms the core/base API can be created with Java.	Websites, Java Beans, Servlets and more powerful server applications can be made with J2EE.
3	One requires only a JVM to use Java SE.	Besides the JVM one requires a Java EE compatible application server like Glassfish, JBoss and others.
4	J2SE has been renamed as JSE.	J2EE has been renamed as JEE.
5	Java for the Workstation/desktop	Java for the server.



What is J2EE?

- An enterprise is a business organization, and enterprise applications are application software that facilitate various activities in an enterprise.
- J2EE defines a model for developing multi-tier, web based, enterprise applications with distributed components.
- Formally, J2EE is a public specification that embodies several technologies like JavaMail, Activation, JAXB (Java API for XML Binding), Servlets, JSF (Java Server Faces), JMS (Java Messaging Service), EJB (Enterprise Java Beans), and others.
- The basic idea behind the J2EE platform released in early 2000 is to provide a simple, unified standard for distributed applications through a component-based application model.

Why J2EE?

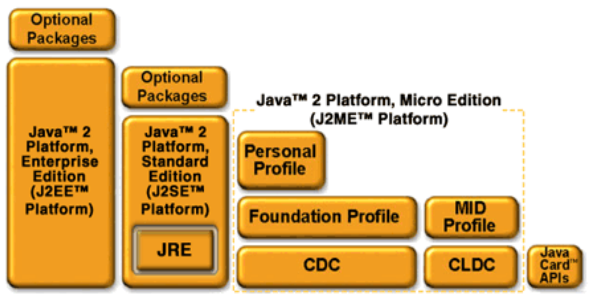
J2EE Provides:

- Enabling technology
- Standards based application model
- A common architecture that provides key common functionality:
  - Security
  - Integration
  - Scalability
  - Programming Productivity
  - Reliability and Availability

J2EE Technologies

- Java Servlets
- JSP
- EJB
- JMS
- JDBC
- JNDI
- JTA / JTS
- JavaMail
- JAAS
- JAXP
- JAF
- JCA

The Java 2 Platform


Optional Packages
Java™ 2 Platform, Enterprise Edition (J2EE™ Platform)
Optional Packages
Java™ 2 Platform, Standard Edition (J2SE™ Platform)
JRE
Java™ 2 Platform, Micro Edition (J2ME™ Platform)
Personal Profile
Foundation Profile
MID Profile
CDC
CLDC
Java Card™ APIs

Introduction to CGI

- When it first emerged, the Internet consisted of only static contents written using Hypertext Markup Language (HTML).
- Dynamic web contents were made possible through the Common Gateway Interface (CGI) technology.
- CGI enables the web server to call an external program and pass HTTP request information to that external program to process the request.
- The response from the external program is then passed back to the web server, which forwards it to the client browser.

What is CGI?

- CGI is a way to interface programs, such as search engines, with Web servers.
- HTTP (Web) servers are designed primarily to serve up HTML documents. But CGI files are not documents they are programs. Therefore, to store CGI programs most Web servers use a special directory, commonly named cgi-bin.
- The Web server knows that files stored in the cgi-bin directory are to be executed rather than simply sent to the user's Web browser for display.
- CGI programs can be written in a wide variety of languages, including DOS batch files, BASIC, C, and scripting languages such as Perl.

Which program is a CGI program?

- A program must meet the following criteria to qualify as a CGI program

i) One should be able to type it directly by typing its name from the command line. A java program does not qualify to be a CGI program because it cannot be executed in the Java Virtual Machine unless we type "java program-name" in the command prompt.

ii) The program should generate a valid content-type header.

What can CGI programs not do?

A CGI program would not do the following:

- It does not interact with a user directly.
- It does not interact directly with a web browser or a graphical user interface. In other words it does not display or retrieve information from menus, commands and other interactive features of a client browser.
- It does not create graphics or windows by itself.

Introduction to CGI-Perl Interface

- CGI programs can be written in any language that can be called by the web server.
- Over the course of time, Perl became the most popular language to write CGI programs. Together the interface was popularly called the CGI-Perl Interface.

Advantages of CGI Programs

- CGI programs are portable and work on a wide variety of web servers and platforms.
- They are language independent and can be written in any language and work in a wide variety of environments. Some of the programming languages used to write CGI scripts are Perl, UNIX shell, C language, Visual Basic, Python, C# and Java.
- They provide simple interfaces for the client to interact with the web servers.
- They are scalable programs used to perform simple tasks in the application layer as well as more complex tasks such as interacting with databases and shopping carts.
- The provide interactivity to a web application and enhance user experience.
- CGI programs are cost effective. By using them businesses can lower their development and maintenance costs.

Disadvantages / Problems with CGI/Perl Interface

- As the number of users visiting a popular web site increased exponentially, CGI failed to deliver scalable Internet applications.
- CGI programs are memory-intensive. Every time a request is made to a server, it launches the CGI program. If written in a scripting language, the interpreter for the script evaluates the entire script to execute the CGI program. With busy server traffic, repeated requests will consume a great deal of server resources.
- CGI programs are not easy to write. They require complex programming and designing skills on the part of the web developers since a lot depends upon how they are implemented in the server environment.
- If proper care is not taken, CGI programs may compromise server security.
- With most CGI programs being well-known, free and easily available. Their strengths and vulnerabilities are known to most web-developers. This can result in exploitation and mis-use.

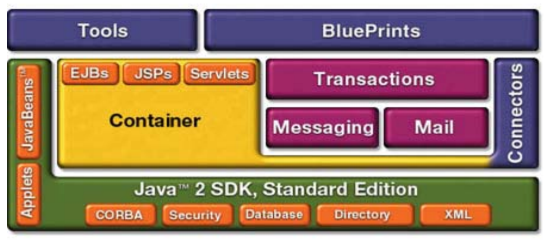
Alternatives to CGI-Perl Interface

- Gradually, new and better technologies have replaced CGI as the main technology for web application development. Some examples are:
- **ColdFusion:** Allaire's ColdFusion provided HTML-like custom tags that can be used to perform a number of operations, especially querying a database.
- **Server-side JavaScript (SSJS):** An extension of the JavaScript language, the scripting language for client-side web programming. SSJS can access Java classes deployed at the server side using the LiveWire technology from Netscape.
- **PHP:** An exciting open-source technology that has matured in recent years. The technology provides easy web application development with its session management and includes some built-in functionality, such as file upload.

Other Alternative Technologies

- **Servlet:** The servlet technology was introduced by Sun Microsystems in 1996.
- **Java Server Pages (JSP):** An extension of the servlet technology.
- **Active Server Pages (ASP):** Microsoft's ASP employs scripting technologies that work in Windows platforms. Windows ASP works with the Internet Information Server web server. This technology will soon be replaced by Active Server Pages.NET.
- **Active Server Pages.NET (ASP.NET):** Part of Microsoft's .NET initiative. Interestingly, the .NET Framework employs a runtime called the Common Language Runtime that is very similar to Java Virtual Machine and provides a vast class library available to all .NET languages and from ASP.NET pages.

J2EE Components



http://java.sun.com/j2ee/overview3.html

Developing Web Aplications

- Two main architectures when developing web applications in Java.
- The first architecture utilizes servlets and JSP in the middle tier to serve clients and process the business logic.

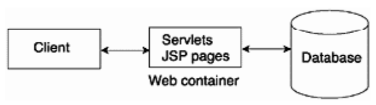


Fig: Servlet/ JSP Application Architecture
Small to medium-size applications use this design model

- The second architecture includes the use of J2EE server and Enterprise JavaBeans (EJB) and this is especially useful for large enterprise applications that need scalability.

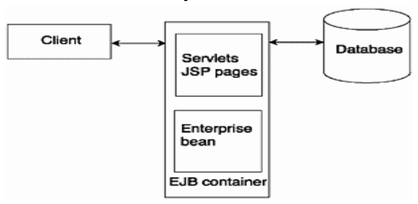


Fig: J2EE Application Architecture

J2EE Architecture

- **Components**
  - Enterprise Java Beans (EJB)
  - Java Server Pages (JSP) } Web Component
  - Servlets
- **Containers** (service providers): J2EE container is a runtime to manage application components developed according to the API specifications and to provide access to the J2EE APIs. Ex: Applet Container, EJB Container, Web Container (for servlets and JSP), Application Client Containers.
- **Connectors** (connection service providers)

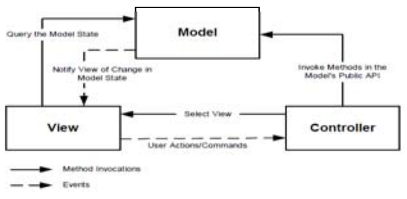
Developing J2EE Applications

- The J2EE specification specifies the following steps in application development and deployment process:

1. Application component development.
2. Composition of application components into modules.
3. Composition of modules into application.
4. Application deployment.

MVC Architecture

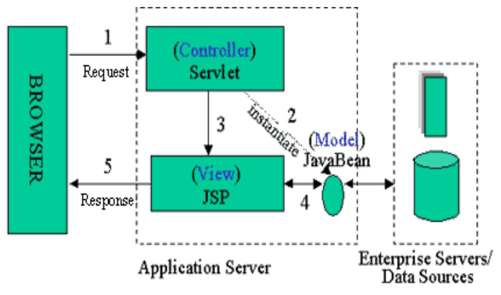
- The MVC (Model-View-Controller) architecture is a way of decomposing an application into three parts: the model, the view and the controller.
- It was originally applied in the graphical user interaction model of input, processing and output.



MVC Architecture

- The Model represents the structure of the data in the application, as well as application-specific operations on those data.
- A View (of which there may be many) presents data in some form to a user, in the context of some application function.
- A Controller translates user actions (mouse motions, keystrokes, words spoken, etc.) and user input into application function calls on the model, and selects the appropriate View based on user preferences and Model state.

MVC Structure for J2EE



MVC and J2EE Applications

- In J2EE applications, MVC architecture is used for separating business layer functionality represented by JavaBeans or EJBs (the model) from the presentation layer functionality represented by JSPs (the view) using an intermediate servlet based controller.
- However, a controller design must accommodate input from various types of clients including HTTP requests from web clients, WML from wireless clients, and XML-based documents from suppliers and business partners.
- For HTTP Request/Response paradigm, incoming HTTP requests are routed to a central controller, which in turn interprets and delegates the request to the appropriate request handlers. This is also referred to as MVC Type-II (Model 2) Architecture.

Details of MVC Design Pattern

- Name (essence of the pattern)
  - Model View Controller MVC
- Context (where does this problem occur)
  - MVC is an architectural pattern that is used when developing interactive application such as a shopping cart on the Internet.
- Problem (definition of the reoccurring difficulty)
  - User interfaces change often, especially on the internet where look-and-feel is a competitive issue. Also, the same information is presented in different ways. The core business logic and data is stable.

MVC continued

- Solution (how do you solve the problem)
  - Use the software engineering principle of "separation of concerns" to divide the application into three areas:

    ✓Model encapsulates the core data and functionality

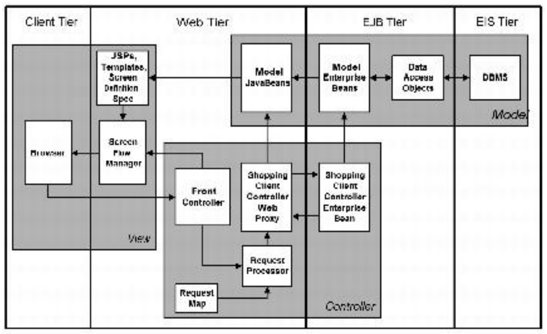
    ✓View encapsulates the presentation of the data there can be many views of the common data

    ✓Controller accepts input from the user and makes request from the model for the data to produce a new view.

Java Pet Store MVC Design Pattern

- The Pet Store application implements MVC (Model-View-Controller) design, and demonstrates one way to design an application that should scale well.
- The Java Pet Store is a reference application that demonstrates J2EE technologies.
- The Java Pet Store design is divided into multiple tiers:
  - A. Client tier
  - B. Web tier
  - C. Enterprise JavaBeans tier
  - D. Enterprise Information System tier.
- These tiers are not necessarily arranged hierarchically.
- Each tier may communicate directly with other tiers, or indirectly by way of intermediate tiers.

J2EE Architecture Tiers

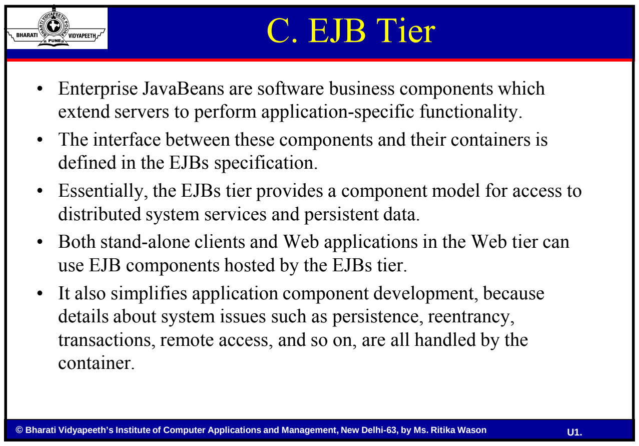


A. Details of Client Tier

- The Client tier is responsible for presenting data to the user, interacting with the user, and communicating with the other tiers of the application.
- The Client tier is the only part the application the user ever sees.
- The Client tier communicates with other tiers by way of well-defined interfaces.
- A separate Client tier in the design provides flexibility and extensibility.
- In The Java Pet Store Client tier consists mainly of a browser displaying Web pages generated from server-side JSP pages in the Web tier.
- Future new clients can be written using technologies or languages that do not yet even exist, since they must conform only to the interface for communicating with other tiers

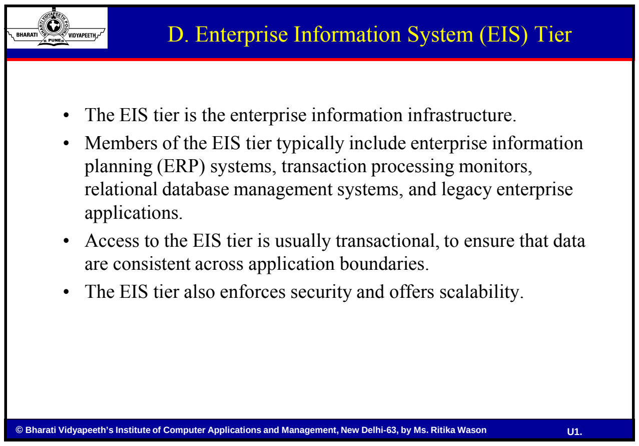
B. Web Tier

- The Web tier is responsible for performing all Web-related processing, such as serving HTML, instantiating Web page templates, and formatting JSP pages for display by browsers.
- The Web tier in the Java Pet Store does all of these, and takes on the Controller functions for the Web application, caching model data interpreting user inputs, selecting appropriate Views based on application flow, and managing database connections.



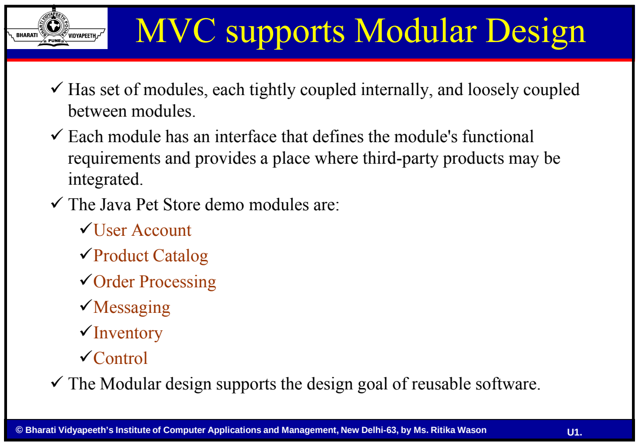
C. EJB Tier

- Enterprise JavaBeans are software business components which extend servers to perform application-specific functionality.
- The interface between these components and their containers is defined in the EJBs specification.
- Essentially, the EJBs tier provides a component model for access to distributed system services and persistent data.
- Both stand-alone clients and Web applications in the Web tier can use EJB components hosted by the EJBs tier.
- It also simplifies application component development, because details about system issues such as persistence, reentrancy, transactions, remote access, and so on, are all handled by the container.



D. Enterprise Information System (EIS) Tier

- The EIS tier is the enterprise information infrastructure.
- Members of the EIS tier typically include enterprise information planning (ERP) systems, transaction processing monitors, relational database management systems, and legacy enterprise applications.
- Access to the EIS tier is usually transactional, to ensure that data are consistent across application boundaries.
- The EIS tier also enforces security and offers scalability.



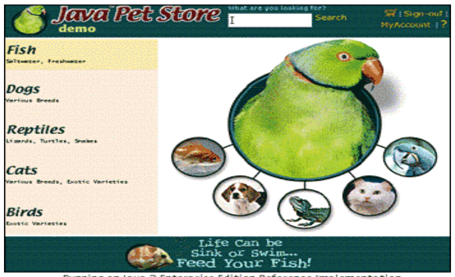
MVC supports Modular Design

- ✓ Has set of modules, each tightly coupled internally, and loosely coupled between modules.
- ✓ Each module has an interface that defines the module's functional requirements and provides a place where third-party products may be integrated.
- ✓ The Java Pet Store demo modules are:
  - ✓User Account
  - ✓Product Catalog
  - ✓Order Processing
  - ✓Messaging
  - ✓Inventory
  - ✓Control
- ✓ The Modular design supports the design goal of reusable software.

Java Pet store- MVC

- Views
  - JSP pages, composed with templates and displayed in an HTML browser
- Controller
  - maps user input from the browser to request events, and forwards those events to the Shopping Client Controller in the EJB tier.
- Model
  - EJB Tier

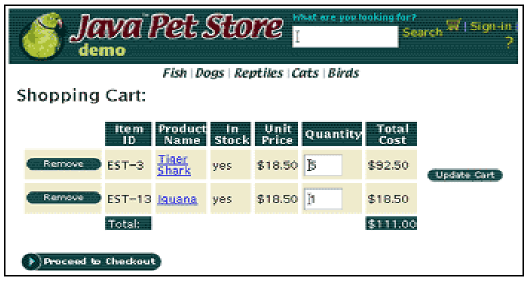
JSP Example



Running on Java 2 Enterprise Edition Reference Implementation

ShoppingCart.jsp

- Java Server Pages (JSP)


Java Pet Store
demo
What are you looking for?
Search
Sign-in
?
Fish | Dogs | Reptiles | Cats | Birds

Shopping Cart:

	Item ID	Product Name	In Stock	Unit Price	Quantity	Total Cost	
Remove	EST-3	Tiger Shark	yes	$18.50	5	$92.50	Update Cart
Remove	EST-13	Iguana	yes	$18.50	1	$18.50	
	Total:					$111.00	

Advantages of MVC

- Separating Model from View (that is, separating data representation from presentation)

  - easy to add multiple data presentations for the same data,

  -facilitates adding new types of data presentation as technology develops.

  -Model and View components can vary independently enhancing maintainability, extensibility, and testability.
- Separating Controller from View (application behavior from presentation)

  - permits run-time selection of appropriate Views based on workflow, user preferences, or Model state.

Advantages of MVC design Pattern

- Separating Controller from Model (application behavior from data representation)

  - allows configurable mapping of user actions on the Controller to application functions on the Model.

Consequences/ Benefits

- We make changes without bringing down the server.
- We leave the core code alone
- We can have multiple versions of the same data displayed
- We can test our changes in the actual environment.
- We have achieved "separation of concerns"

What is a Servlet?

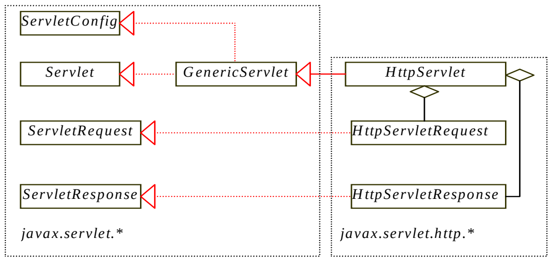
- Servlet is a Java technology based web component, managed by a container, that generates dynamic content.
- Servlet can be considered as a tiny Java program which processes user request and generates dynamic content.
- There are many other alternatives like PHP, ASP.NET or CGI, for developing dynamic web sites. Benefit of using servlets over other technologies is that servlets are developed in Java, so they come with all benefits of Java language and are platform independent.
- Following are the some advantages of using servlets.
  (1) They are generally much faster than CGI scripts.
  (2) They use a standard API that is supported by many web servers.
  (3) They have all the advantages of the Java programming language, including ease of development and platform independence.
  (4) They can access the large set of APIs available for the Java platform.

What is a Servlet Container?

Servlet container (also known as servlet engine) is a runtime environment, which implements servlet API and manages life cycle of servlet components.
Container is responsible for instantiating, invoking, and destroying servlet components.
One example of container is Apache Tomcat which is an opensource container.

The Servlet API

- The servlet API provides the interfaces and classes that support servlets. These interfaces and classes are grouped into two packages: javax.servlet, and javax.servlet.http.



Servlet Life Cycle

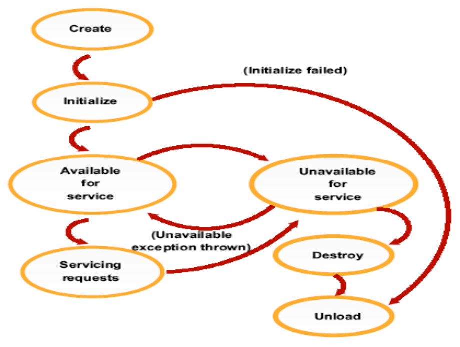
- The life cycle of a servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps.
- If an instance of the servlet does not exist, the Web container
  - Loads the servlet class.
  - Creates an instance of the servlet class.
  - Initializes the servlet instance by calling the init method. Initialization is covered in Initializing a Servlet.
- Invokes the service method, passing a request and response object.
- If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.

Servlet Life Cycle (Contd.)

The life cycle of a servlet can be categorized into four parts:

- **Loading and Instantiation:** The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the attribute <load-on-startup> of web.xml file. If the attribute <load-on-startup> has a positive value then the servlet is loaded with loading of the container otherwise it loads when the first request comes for service. After loading of the servlet, the container creates the instances of the servlet.
- **Initialization:** After creating the instances, the servlet container calls the init() method and passes the servlet initialization parameters to the init() method. The init() must be called by the servlet container before the servlet can service any request. The initialization parameters persist untill the servlet is destroyed. The init() method is called only once throughout the life cycle of the servlet.
  The servlet will be available for service if it is loaded successfully otherwise the servlet container unloads the servlet.
- **Servicing the Request:** After successfully completing the initialization process, the servlet will be available for service. Servlet creates seperate threads for each request. The sevlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet() or doPost()) for handling the request and sends response to the client using the methods of the response object.
- **Destroying the Servlet:** If the servlet is no longer needed for servicing any request, the servlet container calls the destroy() method . Like the init() method this method is also called only once throughout the life cycle of the servlet. Calling the destroy() method indicates to the servlet container not to sent the any request for service and the servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection.

Servlet Life Cycle



Points to Remember

- init
- Executed once when the servlet is first loaded.
- *Not called for each request.*
- Service
- Called in a new thread by server for each request.
- Dispatches to doGet, doPost, etc.
- Do not override this method!
- doGet, doPost, do*Blah*
- Handles GET, POST, etc. requests.
- Override these to provide desired behavior.
- destroy
- Called when server deletes servlet instance.
- *Not called after each request.*

Why not to Override service()

- The service method does other things besides just calling doGet
- You can add support for other services later by adding doPut, doTrace, etc.
- You can add support for modification dates by adding a getLastModified method
- The service method gives you automatic support for:
  - HEAD requests
  - OPTIONS requests
  - TRACE requests
- Alternative: have doPost call doGet

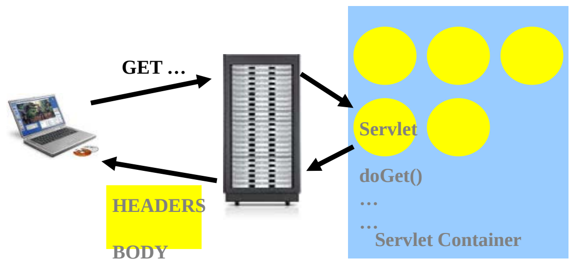
Why Java Servlets Instead of CGI?

- Efficient, Convenient, Powerful, Portable, Secure, Inexpensive
  - Lightweight threads instead of OS threads created
  - Single copy of code brought into memory for all threads versus per thread
  - Data (session state) can be stored across threads within servlet container
  - Java is portable and secure
  - Requires little expense once servlet container integrated with web server

Servlet Structure

- Java Servlet Objects on Server Side
- Managed by Servlet Container
  - Loads/unloads servlets
  - Directs requests to servlets
- Request → doGet()
- Each request is run as its own thread

Web App with Servlets



5 Simple Steps for Java Servlets

1. Subclass off HttpServlet
2. Override doGet(....) method
3. HttpServletRequest
   - getParameter("paramName")
4. HttpServletResponse
   - set Content Type
   - get PrintWriter
   - send text to client via PrintWriter
5. Don't use instance variables

Servlet Lifecycle (Creation)

- Single instance created
- init() method called
- You can override init() in your subclass of HttpServlet to do some initial code....
- init() is NOT called again on further requests
- On each request, the server spawns a new thread and calls service()
- service() checks HTTP request type and calls appropriate doXXXX (Get, Post, Put...) don't override service (unless you really know what you're doing)
- Real meat of the web app is here
- doPost() can call doGet(), or viceversa
- no doHead()... system uses headers of doGet() result

Servlet Lifecycle (destroy())

- For some reason (servlet idle, etc) the server may want to remove the servlet from memory
- destroy() allows you to close DB connections, wrap up, etc...
- Don't count on destroy to write persistent state (server may crash before you ever get here!)

Environment Variables

- JavaServlets do not require you to use the clunky environment variables used in CGI
- Individual functions:

▪ PATH_INFO	req.getPathInfo()
▪ REMOTE_HOST	req.getRemoteHost()
▪ QUERY_STRING	req.getQueryString()
▪ …	

Setting Response Components

- Set status first!
  - setStatus(int)
    HttpServletResponse.SC_OK...
  - sendError(int, String)
  - sendRedirect(String url)
- Set headers
  - setHeader(…)
  - setContentType("text/html")
- Output body
  - PrintWriter out = response.getWriter();
  - out.println("<HTML><HEAD>...")

The Servlet Family

- Fundamentally, the job of a servlet is to process a request and to return a response.
- There are different APIs for manipulating the requests that the servlets are receiving and for generating the response that the servlets are sending.
- Servlets are easy to work with because the container handles many functions.
- The container is also responsible for creating the request and response objects used by methods like doGet() and doPost(). The container also controls the life cycle of a servlet using methods declared in the Servlet interface.

Generic and Http Servlet

- GenericServlet implements the Servlet interface as well as the ServletConfig interface. The service() method is its only abstract method.
- Finally, the HttpServlet abstract class extends GenericServlet and adds methods for dealing with HTTP–specific requests. It has no abstract methods, but if you don't override one of the basic methods it won't have any useful functionality and so is declared to be an abstract class.

The Servlet Interface

- When a user makes a request for a servlet, the container creates an instance on the servlet if one doesn't already exist.
- If there is an init() method, it will be called and must complete successfully before the servlet sees any client requests.
- After the init() method returns, the container may or may not call the service() method one or more times, passing in ServletRequest and ServletResponse objects as arguments.
- Finally, the container can call the destroy() method to finalize the servlet and clean up various resources.

Generic Servlet

- We can create a servlet by extending GenericServlet and overriding the service() method.
- The GenericServlet class provides implementations of all of the methods in the interfaces Servlet and ServletConfig except for service().
- GenericServlet also provides an implementation for the methods of the ServletConfig interface.
- GenericServlet also contains two log() methods not specified in either the Servlet or the ServletConfig interface. The first takes a string as its argument that will be the message to be written to a servlet log file. The message will be tagged with a particular servlets name in order to figure out which message belongs to which servlet. This method is used with various life-cycle methods so that when init() or destroy() is called an appropriate log entry is genetrated.

Generic Servlet

- The second version of the log() method takes an instance of Throwable as its second argument.
- This signature of log() is implemented in GenericServlet to write the message you specify as the first argument and to write a stack trace for the specified exception into the log file.

HttpServlet

- GenericServlet may be directly extended by a servlet, although it is more common to extend a protocol-specific subclass such as HttpServlet.
- HttpServlet provides an abstract class to be subclassed to create an HttpServlet suitable for a Website.
- A subclass of HttpServlet must override atleast one method, usually one of these:

i) doGet() if the servlet supports Http Get requests.
ii) doPost(), for Http Post requests.
iii) doPut(), for Http Put requests.
iv) doDelete() for Http delete requests.
v) init() and destroy() to manage resources that are held for the life of the servlet.

HttpServlet

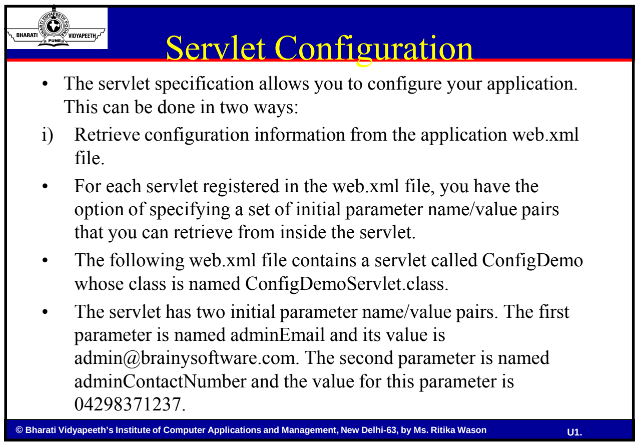
- HttpServlet extends GenericServlet by adding methods to handle HTTP requests.
- GET and POST requests can be handled by using the HttpServlet methods doGet() and doPost().
- The service methods doDelete(), doHead(), doOptions(), doPut(), and doTrace() are also available to handle DELETE, HEAD, OPTIONS, PUT, and TRACE respectively.
- Each of these methods takes an HttpServletRequest object and an HttpServletResponse object as arguments, and can throw a ServletException or an IOException.
- There are no abstract methods in the HttpServlet class. The service() method, which was abstract in the parent class, is no longer abstract in HttpServlet.

HttpServlet

- Nevertheless, HttpServlet is an abstract class, so you can create servlets by extending it and overriding one of the service methods.
- The final method that has been added in HttpServlet is getLastModified(). It can help the client to not reload a page that hasn't changed since the last time he or she accessed it.

Generic versus Http Servlet

S.No	GenericServlet (javax.servlet.GenericServlet)	HttpServlet (javax.servlet.HttpServlet)
1.	Signature: public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable	Signature: public abstract class HttpServlet extends GenericServlet implements java.io.Serializable
2.	GenericServlet defines a generic, protocol-independent servlet.	HttpServlet define a Http protocol specific servlet.
3.	GenericServlet gives a blueprint and makes writing servlets easier.	HttpServlet extends the GenericServlet and hence inherits the properties of GenericServlet.



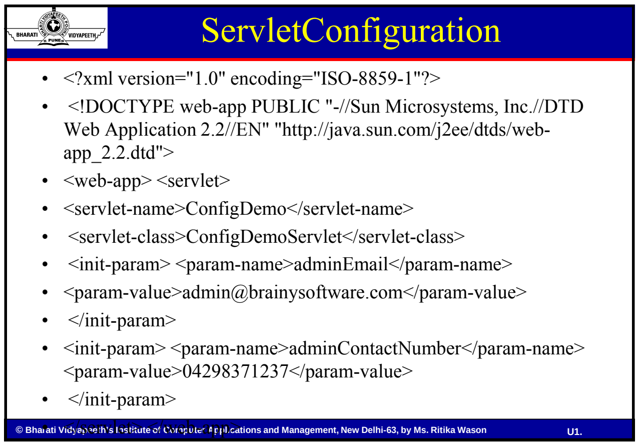
Servlet Configuration

- The servlet specification allows you to configure your application. This can be done in two ways:

i) Retrieve configuration information from the application web.xml file.

- For each servlet registered in the web.xml file, you have the option of specifying a set of initial parameter name/value pairs that you can retrieve from inside the servlet.
- The following web.xml file contains a servlet called ConfigDemo whose class is named ConfigDemoServlet.class.
- The servlet has two initial parameter name/value pairs. The first parameter is named adminEmail and its value is admin@brainysoftware.com. The second parameter is named adminContactNumber and the value for this parameter is 04298371237.

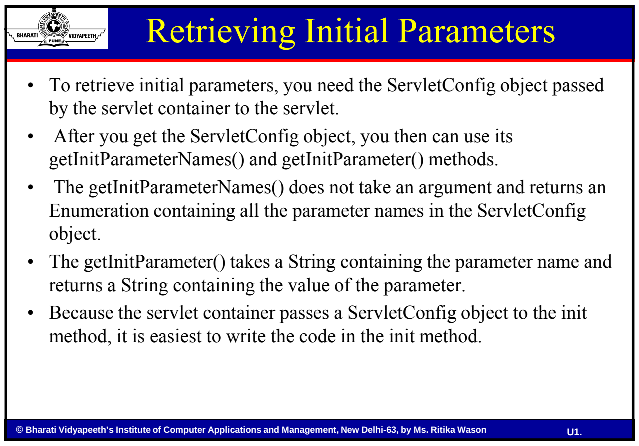
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



ServletConfiguration

- <?xml version="1.0" encoding="ISO-8859-1"?>
- <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
- <web-app> <servlet>
- <servlet-name>ConfigDemo</servlet-name>
- <servlet-class>ConfigDemoServlet</servlet-class>
- <init-param> <param-name>adminEmail</param-name>
- <param-value>admin@brainysoftware.com</param-value>
- </init-param>
- <init-param> <param-name>adminContactNumber</param-name> <param-value>04298371237</param-value>
- </init-param>

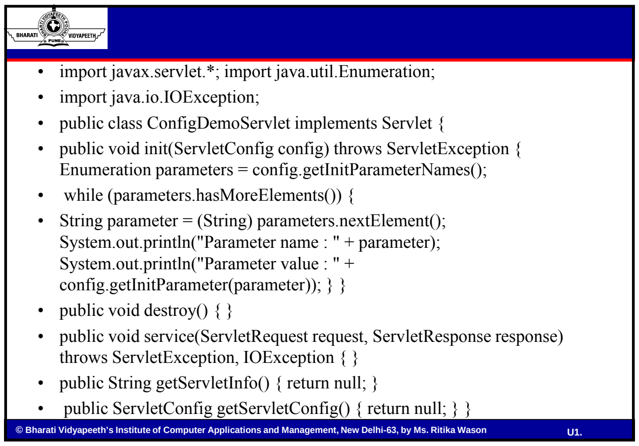
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Retrieving Initial Parameters

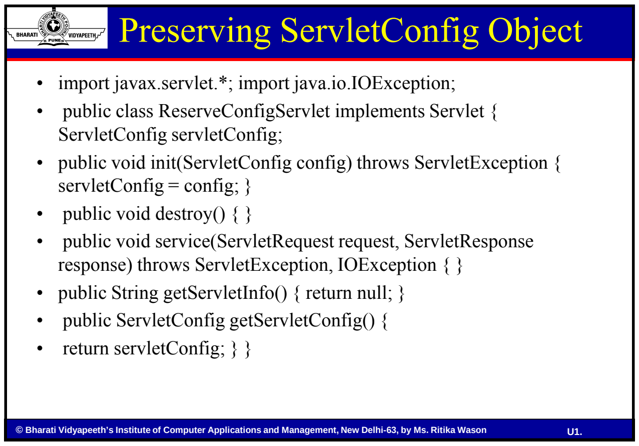
- To retrieve initial parameters, you need the ServletConfig object passed by the servlet container to the servlet.
- After you get the ServletConfig object, you then can use its getInitParameterNames() and getInitParameter() methods.
- The getInitParameterNames() does not take an argument and returns an Enumeration containing all the parameter names in the ServletConfig object.
- The getInitParameter() takes a String containing the parameter name and returns a String containing the value of the parameter.
- Because the servlet container passes a ServletConfig object to the init method, it is easiest to write the code in the init method.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



- import javax.servlet.*; import java.util.Enumeration;
- import java.io.IOException;
- public class ConfigDemoServlet implements Servlet {
- public void init(ServletConfig config) throws ServletException { Enumeration parameters = config.getInitParameterNames();
- while (parameters.hasMoreElements()) {
- String parameter = (String) parameters.nextElement(); System.out.println("Parameter name : " + parameter); System.out.println("Parameter value : " + config.getInitParameter(parameter)); } }
- public void destroy() { }
- public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException { }
- public String getServletInfo() { return null; }
- public ServletConfig getServletConfig() { return null; } }

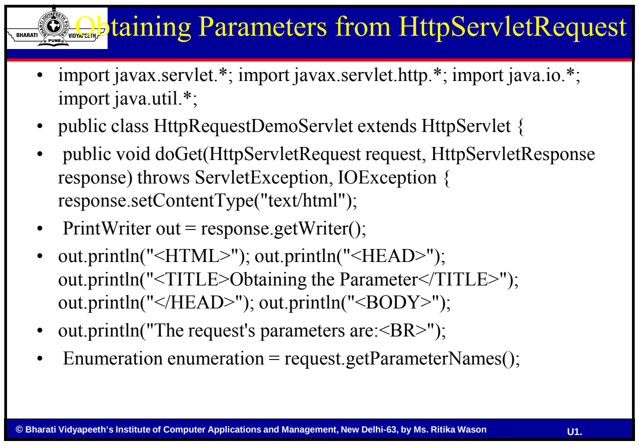
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Preserving ServletConfig Object

- import javax.servlet.*; import java.io.IOException;
- public class ReserveConfigServlet implements Servlet { ServletConfig servletConfig;
- public void init(ServletConfig config) throws ServletException { servletConfig = config; }
- public void destroy() { }
- public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException { }
- public String getServletInfo() { return null; }
- public ServletConfig getServletConfig() {
- return servletConfig; } }

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Obtaining Parameters from HttpServletRequest

- import javax.servlet.*; import javax.servlet.http.*; import java.io.*; import java.util.*;
- public class HttpRequestDemoServlet extends HttpServlet {
- public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- out.println("<HTML>"); out.println("<HEAD>"); out.println("<TITLE>Obtaining the Parameter</TITLE>"); out.println("</HEAD>"); out.println("<BODY>");
- out.println("The request's parameters are:<BR>");
- Enumeration enumeration = request.getParameterNames();

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.

Obtaining Parameters from HttpServletRequest

- while (enumeration.hasMoreElements()){
- String parameterName = (String) enumeration.nextElement(); out.println(parameterName + ": " + request.getParameter(parameterName) + "<BR>"); } out.println("<FORM METHOD=GET>"); out.println("<BR>First Name: <INPUT TYPE=TEXT NAME=FirstName>");
- out.println("<BR>Last Name: <INPUT TYPE=TEXT NAME=LastName>");
- out.println("<BR><INPUT TYPE=SUBMIT VALUE=Submit>");
- out.println("</FORM>");
- out.println("</BODY>"); out.println("</HTML>"); } }

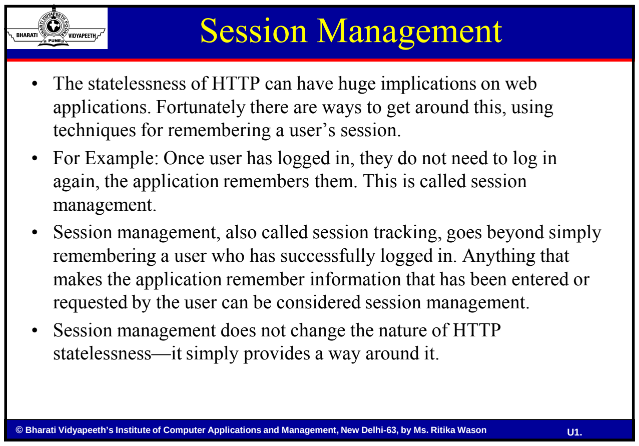
ServletConfig Interface

- public interface ServletConfig :A servlet configuration object used by a servlet container used to pass information to a servlet during initialization.

S.No	Method	Description
1	public java.lang.String **getServletName**()	Returns the name of this servlet instance.
2	public ServletContext **getServletContext**()	Returns a reference to the ServletContext in which the caller is executing.
3	public java.lang.String **getInitParameter**(java.lang.String name)	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist
4	public java.util.Enumeration **getInitParameterNames**()	Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.

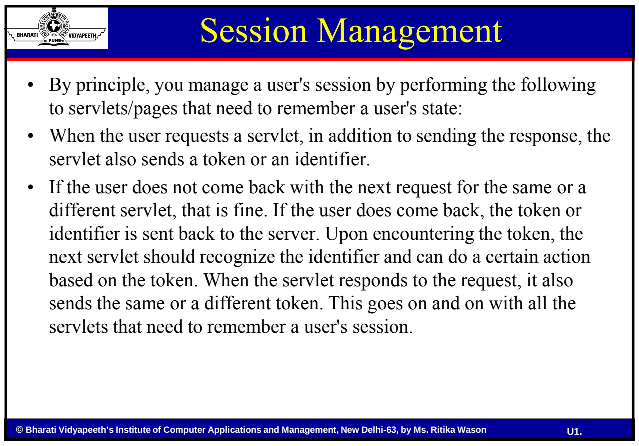
Introduction

- The Hypertext Transfer Protocol (HTTP) is the network protocol that web servers and client browsers use to communicate with each other.
- HTTP is the language of the web. HTTP connections are initiated by a client browser that sends an HTTP request. The web server then responds with an HTTP response and closes the connection.
- If the same client requests another resource from the server, it must open another HTTP connection to the server.
- The server always closes the connection as soon as it sends the response, whether or not the browser user needs some other resource from the server.
- Putting this in a web perspective, because the web server always disconnects after it responds to a request, the web server does not know whether a request comes from a user who has just requested the first page or from a user who has requested nine other pages before. As such, HTTP is said to be stateless.



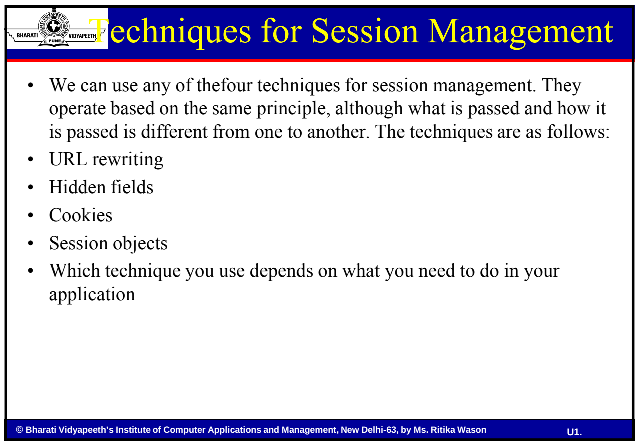
Session Management

- The statelessness of HTTP can have huge implications on web applications. Fortunately there are ways to get around this, using techniques for remembering a user's session.
- For Example: Once user has logged in, they do not need to log in again, the application remembers them. This is called session management.
- Session management, also called session tracking, goes beyond simply remembering a user who has successfully logged in. Anything that makes the application remember information that has been entered or requested by the user can be considered session management.
- Session management does not change the nature of HTTP statelessness—it simply provides a way around it.



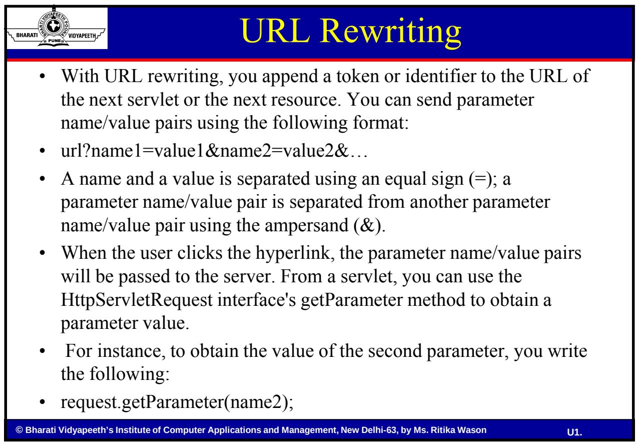
Session Management

- By principle, you manage a user's session by performing the following to servlets/pages that need to remember a user's state:
- When the user requests a servlet, in addition to sending the response, the servlet also sends a token or an identifier.
- If the user does not come back with the next request for the same or a different servlet, that is fine. If the user does come back, the token or identifier is sent back to the server. Upon encountering the token, the next servlet should recognize the identifier and can do a certain action based on the token. When the servlet responds to the request, it also sends the same or a different token. This goes on and on with all the servlets that need to remember a user's session.



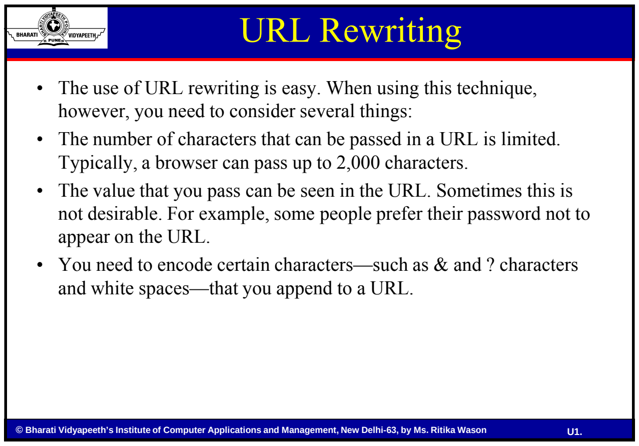
Techniques for Session Management

- We can use any of thefour techniques for session management. They operate based on the same principle, although what is passed and how it is passed is different from one to another. The techniques are as follows:
- URL rewriting
- Hidden fields
- Cookies
- Session objects
- Which technique you use depends on what you need to do in your application



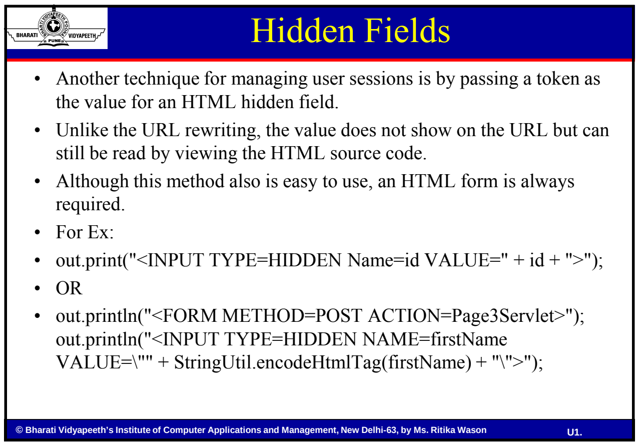
URL Rewriting

- With URL rewriting, you append a token or identifier to the URL of the next servlet or the next resource. You can send parameter name/value pairs using the following format:
- url?name1=value1&name2=value2&…
- A name and a value is separated using an equal sign (=); a parameter name/value pair is separated from another parameter name/value pair using the ampersand (&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a servlet, you can use the HttpServletRequest interface's getParameter method to obtain a parameter value.
- For instance, to obtain the value of the second parameter, you write the following:
- request.getParameter(name2);



URL Rewriting

- The use of URL rewriting is easy. When using this technique, however, you need to consider several things:
- The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters.
- The value that you pass can be seen in the URL. Sometimes this is not desirable. For example, some people prefer their password not to appear on the URL.
- You need to encode certain characters—such as & and ? characters and white spaces—that you append to a URL.



Hidden Fields

- Another technique for managing user sessions is by passing a token as the value for an HTML hidden field.
- Unlike the URL rewriting, the value does not show on the URL but can still be read by viewing the HTML source code.
- Although this method also is easy to use, an HTML form is always required.
- For Ex:
- out.print("<INPUT TYPE=HIDDEN Name=id VALUE=" + id + ">");
- OR
- out.println("<FORM METHOD=POST ACTION=Page3Servlet>"); out.println("<INPUT TYPE=HIDDEN NAME=firstName VALUE=\"" + StringUtil.encodeHtmlTag(firstName) + "\">");

Cookies

- The third technique that you can use to manage user sessions is by using cookies.
- A cookie is a small piece of information that is passed back and forth in the HTTP request and response. Even though a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet.
- The cookie sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.
- Cookies were first specified by Netscape and are now part of the Internet standard as specified in RFC 2109: The HTTP State Management Mechanism. Cookies are transferred to and from the client in the HTTP headers.
- In servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package.

Cookies

- You can create a cookie by calling the Cookie class constructor and passing two String objects: the name and value of the cookie. For instance, the following code creates a cookie object called c1. The cookie has the name "myCookie" and a value of "secret":
- Cookie c1 = new Cookie("myCookie", "secret");
- You then can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:
- response.addCookie(c1);
- Note that because cookies are carried in the request and response headers, you must not add a cookie after an output has been written to the HttpServletResponse object. Otherwise, an exception will be thrown.

Session Objects

- The Session object, represented by the javax.servlet.http.HttpSession interface, is the easiest to use and the most powerful technique for session management.
- For each user, the servlet can create an HttpSession object that is associated with that user only and can only be accessed by that particular user.
- The HttpSession object acts like a Hashtable into which you can store any number of key/object pairs. The HttpSession object is accessible from other servlets in the same application. To retrieve an object previously stored, you need only to pass the key.
- An HttpSession object uses a cookie or URL rewriting to send a token to the client. If cookies are used to convey session identifiers, the client browsers are required to accept cookies.
- Unlike previous techniques, however, the server does not send any value. What it sends is simply a unique number called the session identifier. This session identifier is used to associate a user with a Session object in the server. Therefore, if there are 10 simultaneous users, 10 Session objects will be created in the server and each user can access only his or her own HttpSession object.

Session Object

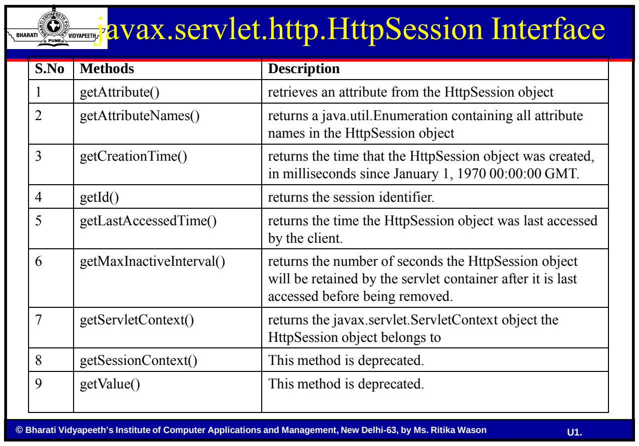
- There are four steps in session tracking using the HttpSession object:
- An HttpSession object is created by a servlet. A session identifier is generated for this HttpSession object by the servlet container. This session identifier will be a random number that is guaranteed to be unique. The HttpSession object then is stored in the server and is associated with the generated session identifier. Also the programmer can store values immediately after creating an HttpSession.
- In the response, the servlet sends the session identifier to the client browser.
- When the client browser requests another resource in the same application,, the session identifier is sent back to the server and passed to next Servlet in the javax.servlet.http.HttpServletRequest object.
- For Servlet2 to have access to the HttpSession object for this particular client, it uses the getSession method of the javax.servlet.http.HttpServletRequest interface. This method automatically retrieves the session identifier from the request and obtains the HttpSession object associated with the session identifier.

Session Object

- Note
- What if the user never comes back after an HttpSession object is created? Then the servlet container waits for a certain period of time and removes that HttpSession object.
- One worry about using Session objects is scalability. In some servlet containers, Session objects are stored in memory, and as the number of users exceeds a certain limit, the server eventually runs out of memory.
- One solution to this memory problem when using Session objects is to save Session objects to the database or disk. However, the Servlet 2.3 Specification does not clearly state how a servlet container should do this.
- The getSession method of the javax.servlet.http.HttpServletRequest interface has two overloads. They are as follows:
- HttpSession getSession()
- HttpSession getSession(boolean create)

Session Management

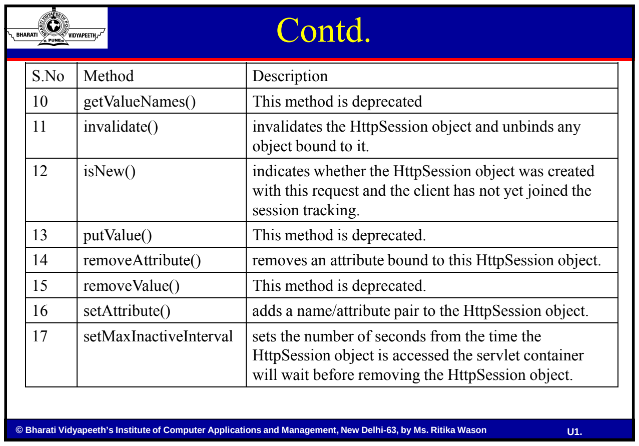
- The first overload returns the current session associated with this request, or if the request does not have a session identifier, it creates a new one.
- The second overload returns the HttpSession object associated with this request if there is a valid session identifier in the request. If no valid session identifier is found in the request, whether a new HttpSession object is created depends on the create value. If the value is true, a new HttpSession object is created if no valid session identifier is found in the request. Otherwise, the getSession method will return null.



javax.servlet.http.HttpSession Interface

S.No	Methods	Description
1	getAttribute()	retrieves an attribute from the HttpSession object
2	getAttributeNames()	returns a java.util.Enumeration containing all attribute names in the HttpSession object
3	getCreationTime()	returns the time that the HttpSession object was created, in milliseconds since January 1, 1970 00:00:00 GMT.
4	getId()	returns the session identifier.
5	getLastAccessedTime()	returns the time the HttpSession object was last accessed by the client.
6	getMaxInactiveInterval()	returns the number of seconds the HttpSession object will be retained by the servlet container after it is last accessed before being removed.
7	getServletContext()	returns the javax.servlet.ServletContext object the HttpSession object belongs to
8	getSessionContext()	This method is deprecated.
9	getValue()	This method is deprecated.

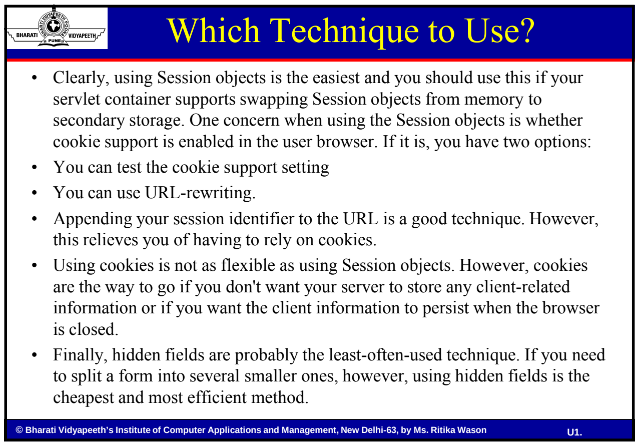
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Contd.

S.No	Method	Description
10	getValueNames()	This method is deprecated
11	invalidate()	invalidates the HttpSession object and unbinds any object bound to it.
12	isNew()	indicates whether the HttpSession object was created with this request and the client has not yet joined the session tracking.
13	putValue()	This method is deprecated.
14	removeAttribute()	removes an attribute bound to this HttpSession object.
15	removeValue()	This method is deprecated.
16	setAttribute()	adds a name/attribute pair to the HttpSession object.
17	setMaxInactiveInterval	sets the number of seconds from the time the HttpSession object is accessed the servlet container will wait before removing the HttpSession object.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Which Technique to Use?

- Clearly, using Session objects is the easiest and you should use this if your servlet container supports swapping Session objects from memory to secondary storage. One concern when using the Session objects is whether cookie support is enabled in the user browser. If it is, you have two options:
- You can test the cookie support setting
- You can use URL-rewriting.
- Appending your session identifier to the URL is a good technique. However, this relieves you of having to rely on cookies.
- Using cookies is not as flexible as using Session objects. However, cookies are the way to go if you don't want your server to store any client-related information or if you want the client information to persist when the browser is closed.
- Finally, hidden fields are probably the least-often-used technique. If you need to split a form into several smaller ones, however, using hidden fields is the cheapest and most efficient method.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.

ServletContext

- In servlet programming, the servlet context is the environment where the servlet runs.
- The servlet container creates a ServletContext object that you can use to access information about the servlet's environment.
- One can obtain a ServletContext object indirectly, from the ServletConfig object passed by the servlet container to the servlet's init method.
- The ServletConfig interface has a method called getServletContext that returns the ServletContext object. You then can use the ServletContext interface's various methods to get the information you need.

ServletContext Methods

S.No	Method	Description
1	getMajorVersion()	Returns an integer representing the major version for the servlet API that the servlet container supports. If the servlet container supports the servlet API version 2.3, this method will return 2.
2	getMinorVersion()	Returns an integer representing the minor version of the servlet API that the servlet container supports. For the servlet API version 2.3, this method will return 3.
3	getAttributeNames()	Returns an enumeration of strings representing the names of the attributes currently stored in the ServletContext.
4	getAttribute()	Accepts a String containing the attribute name and returns the object bound to that name.
5	setAttribute()	Stores an object in the ServletContext and binds the object to the given name. If the name already exists in the ServletContext, the old bound object will be replaced by the object passed to this method.
6	removeAttribute()	Removes from the ServletContext the object bound to a name. The removeAttribute () accepts one argument: the name of the attribute to be removed.

ServletContext Example

- package ServletContextEx; import java.io.*;
- import java.util.Enumeration; import javax.servlet.*;
- import javax.servlet.http.
- public class ContextInitParameter extends HttpServlet implements Servlet { public ContextInitParameter() {}
- protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {doPost(request, response);}
- protected void doPost(HttpServletRequest request,
- HttpServletResponse response) throws ServletException, IOException {response.setContentType("text/html");
- PrintWriter writer = response.getWriter();
- // Get an instance of ServletContext

ServletContext Example (Contd.)

- ServletContext context = getServletContext();

// To read context initialization parameter we can call and pass context.getInitParameter()

// the name of initialization parameter to be read. If the named //parameter does not exists the returned value will be null. In this //example we read an initialization parameter called LOG.PATH

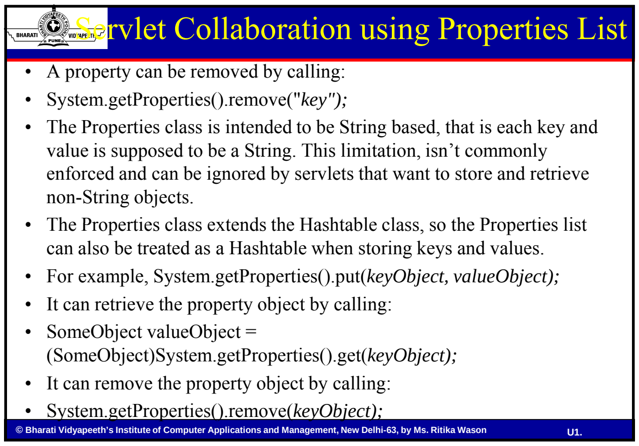
```
•     String logPath = context.getInitParameter("LOG.PATH");
•     writer.println("Log Path: " + logPath + "<br/>");
•       Enumeration enumeration = context.getInitParameterNames();
•     while (enumeration.hasMoreElements()) {
•         String paramName = (String) enumeration.nextElement();
•         String paramValue = context.getInitParameter(paramName);
•         writer.println("Context Init Param: [" + paramName + " = " + paramValue +
   "]<br/>");}}}
```

ServletCollaboration

- Sometimes servlets have to cooperate, usually by sharing some information.
- We call communication of this sort servlet collaboration.
- Collaborating servlets can pass the shared information directly from one servlet to another through method invocations, as shown earlier.
- This approach requires each servlet to know the other servlets with which it is collaborating—an unnecessary burden.

Collaboration through the System Properties List

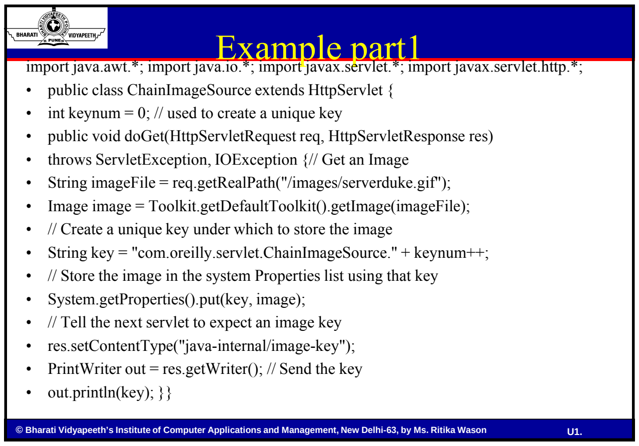
- One simple way for servlets to share information is by using Java's system-wide Properties list, found in the java.lang.System class.
- This Properties list holds the standard system properties, such as java.version and path separator, but it can also hold application-specific properties.
- Servlets can use the properties list to hold the information they need to share.
- A servlet can add (or change) a property by calling:
- System.getProperties().put("*key", "value");*
- That servlet, or another servlet running in the same JVM, can later get the value of the property by calling:
- String value = System.getProperty("*key");*



Servlet Collaboration using Properties List

- A property can be removed by calling:
- System.getProperties().remove("*key");*
- The Properties class is intended to be String based, that is each key and value is supposed to be a String. This limitation, isn't commonly enforced and can be ignored by servlets that want to store and retrieve non-String objects.
- The Properties class extends the Hashtable class, so the Properties list can also be treated as a Hashtable when storing keys and values.
- For example, System.getProperties().put(*keyObject, valueObject);*
- It can retrieve the property object by calling:
- SomeObject valueObject = (SomeObject)System.getProperties().get(*keyObject);*
- It can remove the property object by calling:
- System.getProperties().remove(*keyObject);*

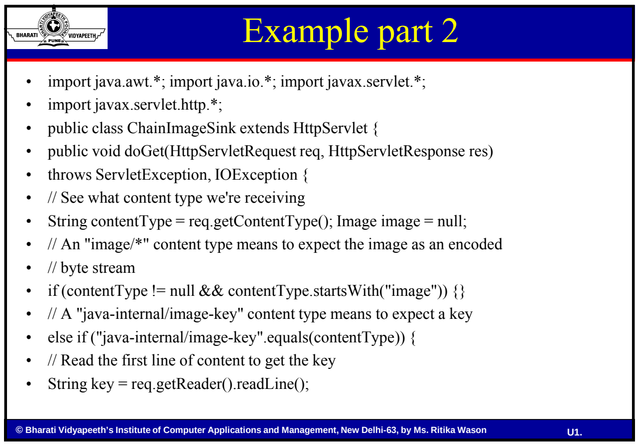
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Example part1

import java.awt.*; import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
- public class ChainImageSource extends HttpServlet {
- int keynum = 0; // used to create a unique key
- public void doGet(HttpServletRequest req, HttpServletResponse res)
- throws ServletException, IOException {// Get an Image
- String imageFile = req.getRealPath("/images/serverduke.gif");
- Image image = Toolkit.getDefaultToolkit().getImage(imageFile);
- // Create a unique key under which to store the image
- String key = "com.oreilly.servlet.ChainImageSource." + keynum++;
- // Store the image in the system Properties list using that key
- System.getProperties().put(key, image);
- // Tell the next servlet to expect an image key
- res.setContentType("java-internal/image-key");
- PrintWriter out = res.getWriter(); // Send the key
- out.println(key); } }

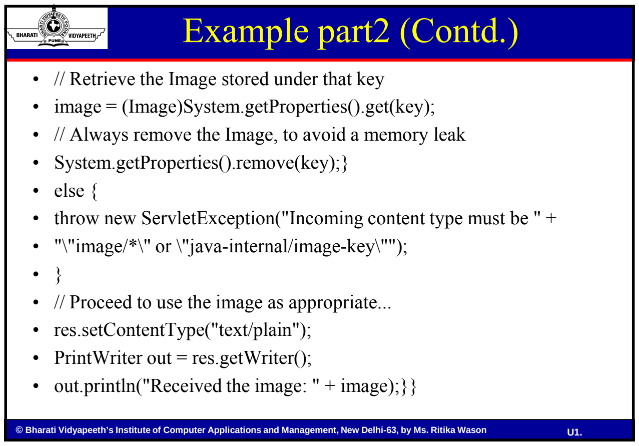
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



Example part 2

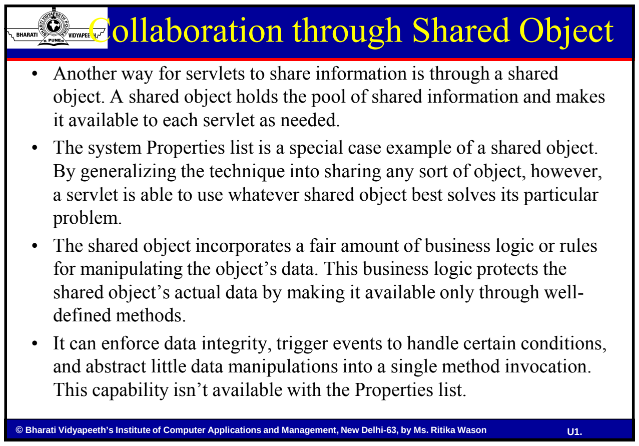
- import java.awt.*; import java.io.*; import javax.servlet.*;
- import javax.servlet.http.*;
- public class ChainImageSink extends HttpServlet {
- public void doGet(HttpServletRequest req, HttpServletResponse res)
- throws ServletException, IOException {
- // See what content type we're receiving
- String contentType = req.getContentType(); Image image = null;
- // An "image/*" content type means to expect the image as an encoded
- // byte stream
- if (contentType != null && contentType.startsWith("image")) {}
- // A "java-internal/image-key" content type means to expect a key
- else if ("java-internal/image-key".equals(contentType)) {
- // Read the first line of content to get the key
- String key = req.getReader().readLine();

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Ms. Ritika Wason U1.



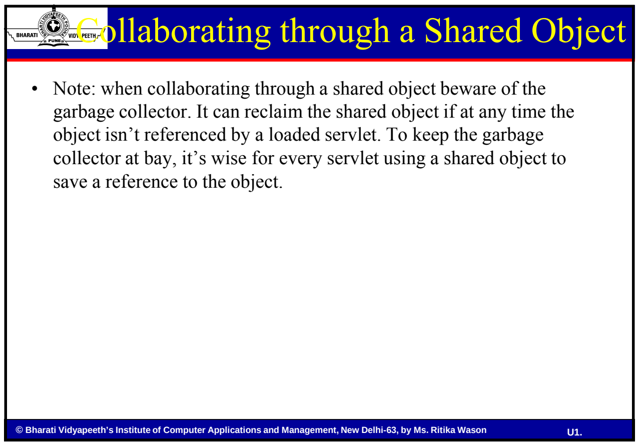
Example part2 (Contd.)

```
• // Retrieve the Image stored under that key
• image = (Image)System.getProperties().get(key);
• // Always remove the Image, to avoid a memory leak
• System.getProperties().remove(key);}
• else {
• throw new ServletException("Incoming content type must be " +
• "\"image/*\" or \"java-internal/image-key\"");
• }
• // Proceed to use the image as appropriate...
• res.setContentType("text/plain");
• PrintWriter out = res.getWriter();
• out.println("Received the image: " + image);}}
```



Collaboration through Shared Object

- Another way for servlets to share information is through a shared object. A shared object holds the pool of shared information and makes it available to each servlet as needed.
- The system Properties list is a special case example of a shared object. By generalizing the technique into sharing any sort of object, however, a servlet is able to use whatever shared object best solves its particular problem.
- The shared object incorporates a fair amount of business logic or rules for manipulating the object's data. This business logic protects the shared object's actual data by making it available only through well-defined methods.
- It can enforce data integrity, trigger events to handle certain conditions, and abstract little data manipulations into a single method invocation. This capability isn't available with the Properties list.



Collaborating through a Shared Object

- Note: when collaborating through a shared object beware of the garbage collector. It can reclaim the shared object if at any time the object isn't referenced by a loaded servlet. To keep the garbage collector at bay, it's wise for every servlet using a shared object to save a reference to the object.

Collaboration through Inheritance

- The easiest technique for servlet collaboration is through inheritance.
- Each servlet interested in collaborating can extend the same class and inherit the same shared information. This simplifies the code for the collaborating servlets and limits access to the shared information to the proper subclasses.
- The common superclass can hold a reference to the shared information, or it can hold the shared information itself.

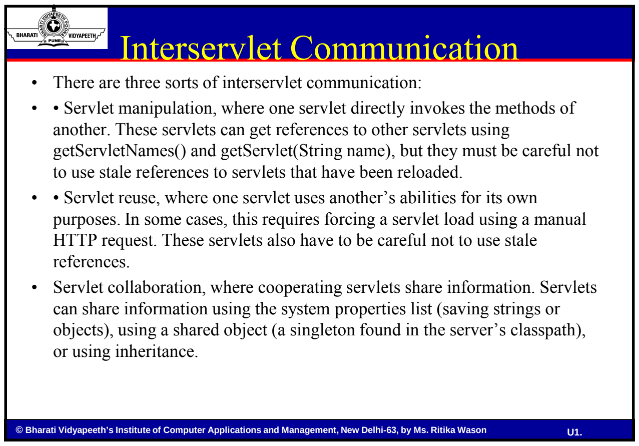
Collaboration through Inheritance (Contd.)

- *Inheriting a shared reference*
- A common superclass can hold any number of references to shared business objects that are easily made available to its subclasses. Example :
- import javax.servlet.*; import javax.servlet.http.*;
- public class BurritoInventorySuperclass extends HttpServlet {
- protected static BurritoInventory inventory = new BurritoInventory();}

The code above shows a superclass. This BurritoInventorySuperclass creates a new BurritoInventory instance. BurritoInventoryProducer and BurritoInventoryConsumer can then subclass BurritoInventorySuperclass and inherit a reference to this instance.

Collaboration through Inheritance (Contd)

- *Inheriting the shared information*
- In addition to holding shared references, a common superclass can hold shared information itself and optionally make it available through inherited business logic methods.



Interservlet Communication

- There are three sorts of interservlet communication:
- • Servlet manipulation, where one servlet directly invokes the methods of another. These servlets can get references to other servlets using getServletNames() and getServlet(String name), but they must be careful not to use stale references to servlets that have been reloaded.
- • Servlet reuse, where one servlet uses another's abilities for its own purposes. In some cases, this requires forcing a servlet load using a manual HTTP request. These servlets also have to be careful not to use stale references.
- Servlet collaboration, where cooperating servlets share information. Servlets can share information using the system properties list (saving strings or objects), using a shared object (a singleton found in the server's classpath), or using inheritance.