**What is Software?**

* Software is the collection of data, programs, procedures, routines and instructions that tell a computer or electronic device how to run, work and execute specific tasks. This is in contrast to hardware, which is the physical system and components that perform the work.

**Types of Software:**

* Application Software
* System Software
* Firmware
* Programming Software
* Driver Software
* Freeware
* Shareware
* Open-Source Software
* Closed Source Software
* Utility Software

**Application Software:**

* Application software is a software program or group of programs designed for end users.
* Word Processing Software: Google Docs, Microsoft Word, WordPad and Notepad
* Database Software: MySQL, Microsoft SQL Server, Microsoft Access, Oracle, IBM DB2 and FoxPro
* Spreading Software: Google Spreadsheets, Apple Numbers and Microsoft Excel
* Multimedia Software: Media Player, Winamp, QuickTime and VLC Media Player
* Introductory Software: Google Slides, Microsoft Powerpoint, Keynotes, Prezzy
* Enterprise Software: customer relationship management (CRM) software (HubSpot, Microsoft Dynamic 365)), project management tools (Jira, Monday), dynamic marketing tools (Marketo, HubSpot), enterprise editing software (ERP) (SAGE, Oracle, Microsoft Dynamics), Wealth Management System (TMS) software (SAP S / 4HANA Finance, Oracle Treasury), Business Intelligence (BI) software (SAP Business Intelligence, MicroStrategy, Microsoft Power BI)
* Information Worker Software: Copywriting tools, resource management tools
* Communication Software: Zoom, Google Meet, Skype
* Educational Software: Dictionaries - Encarta, Britannica; Mathematics: MATLAB; Others: Google Earth, NASA World Wind
* Simulation software: Aircraft and science simulations
* Content Accessing Software: Accessing content through media players, web browsers
* Application Suites: Apache OpenOffice, Microsoft Office365, Apple’s Work, LibreOffice, G-Suite, Oracle E-Business Suite
* Software engineering and product development: IDE or Integrated Development Environments
* Email software: Microsoft Outlook, Gmail, Apple Mail

**System Software:**

* System software provides a platform for other software and includes programs that control the computer itself, such as the computer operating system, file management resources and the disk operating system (or DOS). Program files contain job libraries, system resources, printer drivers and other computer systems, system preferences and other configuration files. System software programs include compilers, compilers, file management tools, system resources and debugging.
* Examples of System Software
* System software runs things in the background and operating systems are an example of system software.
* For desktop computers, laptops and tablets:
* Microsoft Windows
* MacOS (for Apple devices)
* GNU / Linux
* Smartphones:
* Apple iOS
* Google Android
* Windows Phone OS

(Need to plagiarism)

**Firmware:**

* Firmware is software that’s stored on a computer’s motherboard or chipset.
* Its job is to ensure the device works directly. When you switch on your laptop, the Basic Input Output System (BIOS) wakes everything up that available in your PC.

**Programming Software:**

* Programming software, also known as a programming tool or software development tool, is a program that assists or helps software developers or programmers for creating, debugging and maintaining other programs and applications that they are working on.
* Programming software is considered to be a subset of system software
* Examples of Programming Software: - Compilers, assemblers, debuggers, interpreters etc. are examples of programming software. Integrated development environments (IDEs) are combinations of all these software.
  + Most software developers use programming software apps like: GitHub, GitLab, Android Studio, Visual Studio Code, Eclipse, Notepad++

**Driver Software:**

* Driver software communicates with hardware and control devices and peripherals that are attached to a computer. It does this by gathering input from the OS (operating system) and giving instructions to the hardware to perform an action or other designated task.
* Internal components like the hard drive and processor each require its own driver. If a wrong software is installed in the computer then it won’t work perfectly.
* Examples of Driver Software
  + All hardware devices require drivers. For example: Graphic cards, Network cards, Mouse and keyboard

**Freeware**

* Freeware sounds like free software or open-source software but there’s a difference.
* Freeware software does not expose or share its source code. Yet the software owner does not charge others to use it.
* Freeware licenses vary as to what the software can be used for and who can share it.
* Some developers only allow their freeware for private or personal use. Businesses need a paid license or get written permission. An example of this is GPT-3 – and only approved developers and marketers can get access to the program.
* Always read the small print and be wary of the copyright of freeware licenses.
* These softwares are available free of cost. A user can easily download them from the internet and can easily use them without paying any charges or fees. However, they don’t provide any type of liberty to modify the entire software or charging a fixed fee for its distribution.
* A best software development company can develop its own freeware to reach out to more customers. Some of the examples of these software are:
  1. Adobe Reader
  2. Skype
  3. ImgBurn
  4. Audacity
  5. Team Viewer
  6. Yahoo Messenger

**Shareware**

* Like freeware, shareware is free to use and share with others, but only for a short time.
* It acts as an evaluation. You can try some or all of the features before committing to a purchase.
* Examples of Shareware
  + WinZip is one of the most established shareware apps , and some like Adobe Acrobat, PHP Debugger, WinZip, Getright

**Open Source Software**

* Open source means you can explore the actual code that the app was written in.
* Strict software licences restrict what another developer is able to do with the code. However, the ethos behind open-source is to encourage development.
* Open source means evolving the code to make it better for everyone.
* Examples of Open Source Software
* The Linux OS is the perfect example of open-source software.
* Developers can download the source code and edit it as they see fit. New flavours of Linux help target a certain need as a result.
* Such types of software are usually available to users along with their source code which means that the user can easily modify and distribute the software as well as add additional features to them. They can either be chargeable or free. Few of the examples of such software are:

1. Mozilla Firefox
2. Thunderbird
3. GNU Compiler Collection
4. Moodle
5. Apache Web Server

**Closed Source Software**

Utility software is designed to analyses and optimize a device.

These apps are usually bundled with an OS. They track performance and alert the system if there’s a problem like overheating.

Examples of Utility Software

The Windows Task Manager shows all open processes in Windows. It details performance over time and showcases how much memory each uses.

Utilities also include anti-virus software and backup apps.

Benefits of Utility Software

Keeping a close eye on overheating is essential as is scanning for malware.

Utility software helps to keep things stable. It’s designed to make your system run smooth and not get damaged due to overuse.

1. Norton Antivirus
2. McAfee Antivirus
3. WinRAR
4. WinZip
5. Piriform Cleaner
6. Windows File Explorer
7. Directory Opus
8. Razer Cortex

**What is Enterprise Organizations**

Enterprise organizations include:

* Multinational organizations or businesses
* Federal, state, or local government entities
* Medium- to large-scale national companies
* School groups and districts
* Non-profit or charitable organizations spread across multiple areas or regions

**What is Enterprise Applications (what needs do they complete?)**

* Enterprise Applications aim to meet the needs of an enterprise, their functionality must cover a relatively large requirement base.
* Enterprise application software can be broken down into two categories:
* **Software that visualizes, manipulates, and stores a large amount of complex data.** While data warehouses or data analytics software are enterprise solutions, they do not come under the EAS umbrella and are considered separate software.
* **Software that helps in business processes**, ranging from business support to automation.

**What is Enterprise Architecture (EA)**

* An enterprise architecture (EA) is a conceptual blueprint that defines the structure and operation of an organization.
* The intent of an enterprise architecture is to determine how an organization can most effectively achieve its current and future objectives.

**Example of Enterprise Application:**

* Automated billing systems.
* Payment processing.
* Email marketing systems.
* Customer Relationship Management (CRM)
* Enterprise Resource Planning (ERP)
* Business Intelligence (BI)
* Business Continuity Planning (BCP)
* Enterprise Application Integration (EAI)

**How you developed it?**

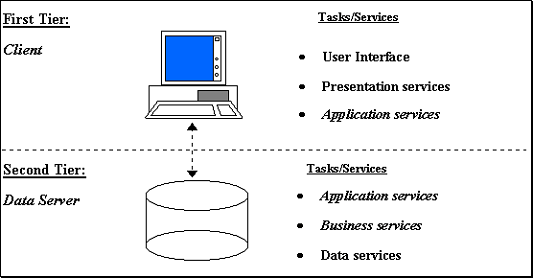
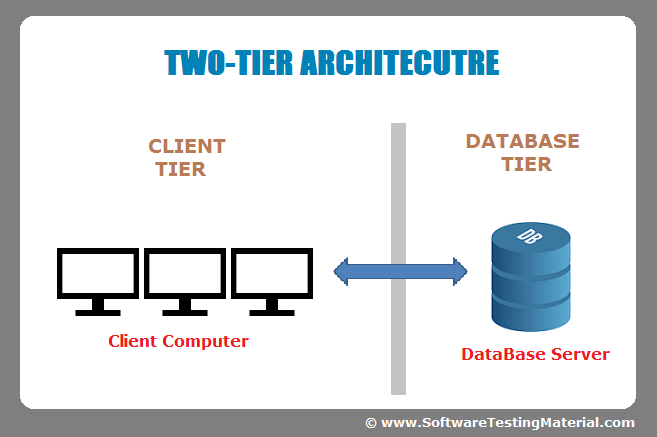
* Enterprise architecture as containing four points-of-view,
  + Business perspective
    - business perspective defines the processes and standards by which the business operates on a day-to-day basis
  + Application perspective,
    - The application perspective defines the interactions among the processes and standards used by the organization Information Perspective
  + The information perspective
    - defines and classifies the raw data (such as document files, databases, images, presentations, and spreadsheets) that the organization requires in order to efficiently operate.
  + Technology perspective.
    - The technology perspective defines the hardware, operating systems, programming, and networking solutions used by the organization.

**Why we developed Enterprise Applications**

* Improved decision making,
* Improved adaptability to changing demands or market conditions,
* Elimination of inefficient and redundant processes, optimization of the use of organizational assets, and minimization of employee turnover.

**Evolution of Enterprise Architecture**

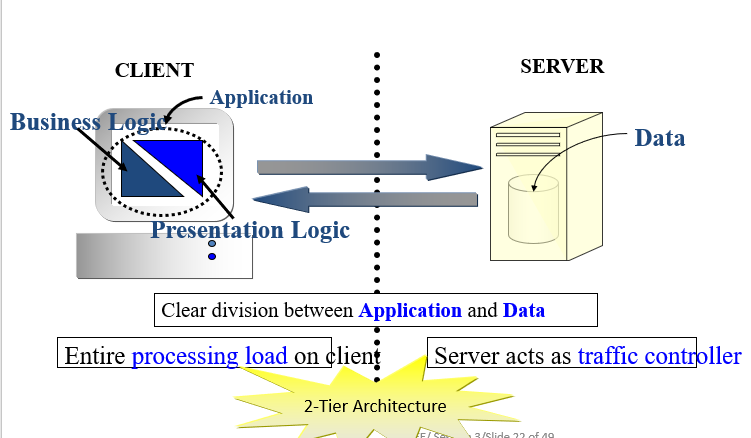
* Mainframe –based Environment
* Client/server computing
  + - Two-Tier Architecture
    - Three -Tier Architecture
    - N-tier Architecture
    - Multi -Tier Architecture
    - Distributed Architecture
    - SOA Architecture







**Client Server Architecture**

****

**Client Server : The Drawbacks**

* Business logic present on each client
* Client waits longer for response
* Business logic + presentation logic bundled together-therefore scalability problems
* Load on server and network as all clients send request to 1 server
* Tiny change to application – entire application has to be changed, and the clients upgraded.

**3-Tier Architecture**

1. Client Tier
2. Middle Tier / Application Tier – receives requests for data, retrieves it, and returns it to client.
3. Data/EIS Tier

**3-Tier Architecture: The benefits**

* Business logic shared between clients; same data returned by data and application tier – presented differently on different clients.
* Business logic can be changed without impacting clients
* Middle tier can be hidden behind firewall
* Maximum use of available system resources
* Parts of architecture can be written without rewriting whole app

**Traditional n-Tier Architecture:**

* Application Logic = Presentation logic + Business Logic
* Infrastructure services provide additional functionalities required by application, such as messaging services and transactional services.

**Traditional n-Tier Architecture: The characteristics:**

* Business logic and presentation logic in same module
* Data base connectivity through same module
* Scalability low
* Business logic difficult to update
* Client and server – stateless communication
* Business logic unaware of different client identities
* Client has to maintain state

**Overcoming the drawbacks – Improving the system:**

* Problem 🡪 Solution
* Middle tier contains one App object 🡪 Extend the middle tier to create one more layer
* For different types of needs different app objects required 🡪 Allow multiple application objects to reside on the server
* Different application objects may not be able to communicate with each other 🡪 Use interfaces to communicate between application objects

**Component n-tier Architecture**

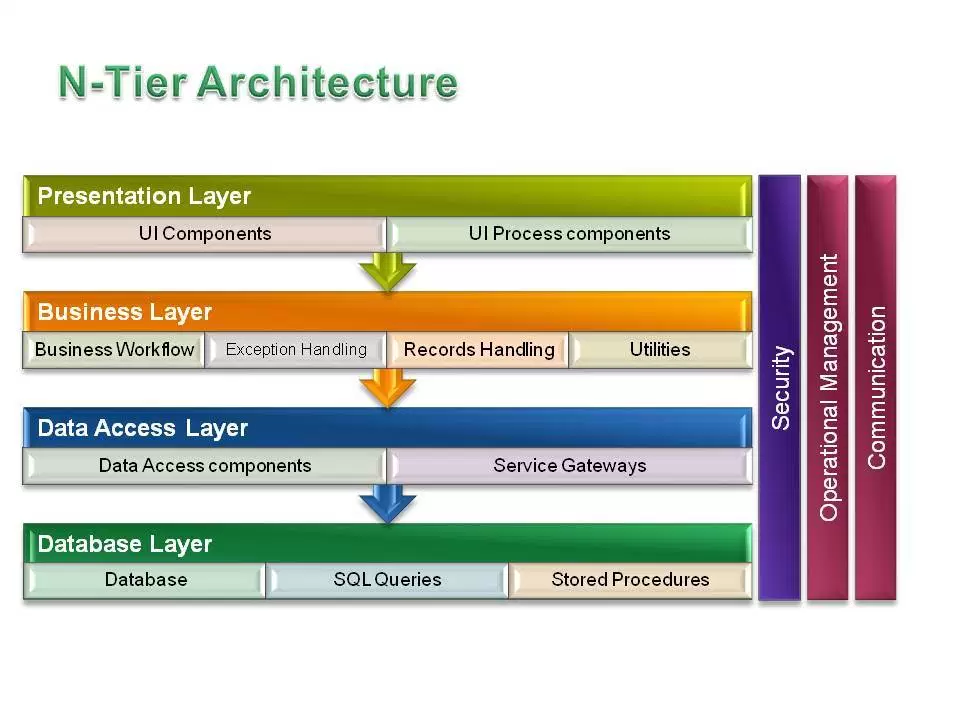
Application object broken into components that can communicate with each other, through interfaces

* Components objects maintain identify and encapsulate remote methods
* Components can be designed to maintain session state on server
* Business logic can be modified without affecting other logic

**Layered Architecture**

* Presentation Layer
* Business Logic Layer
* Database Middleware Layer
* Database Layer
  + JDBC – ODBCC Bridge, perhaps

**N-Tier Architecture**



**Various models of architecture**

* Client-Server
* Traditional n-Tier
* Component-based n-Tier
* Layered

Which architecture would suit which scenario?

* It depends on
  + - Distributed nature of applications
    - Scalability
    - Performance
    - Memory Management

A screenshot of a cell phone

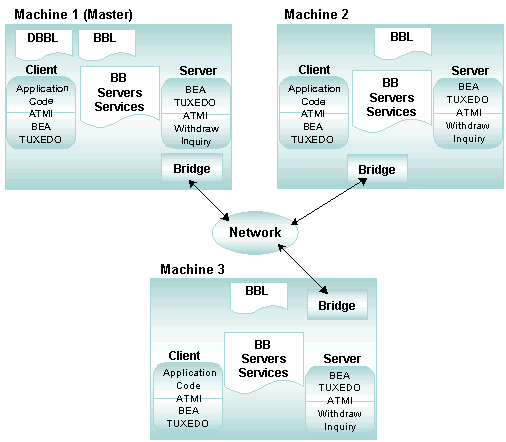
Description generated with high confidenceA screenshot of a cell phone

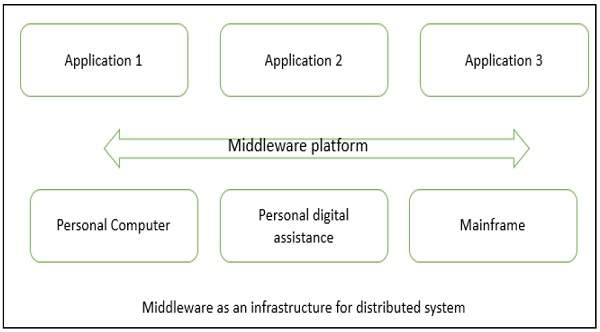
Description generated with very high confidence

**What is Distributed System?**

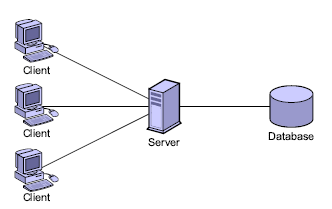
* A distributed system is a network that consists of autonomous computers that are connected using a distribution middleware. They help in sharing different resources and capabilities to provide users with a single and integrated coherent network.

**Sample of a Distributed Application**

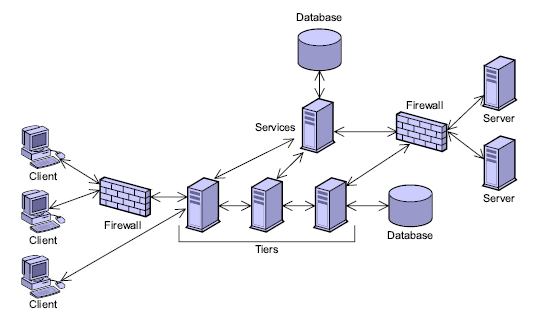




**Minimum Distributed Systems**



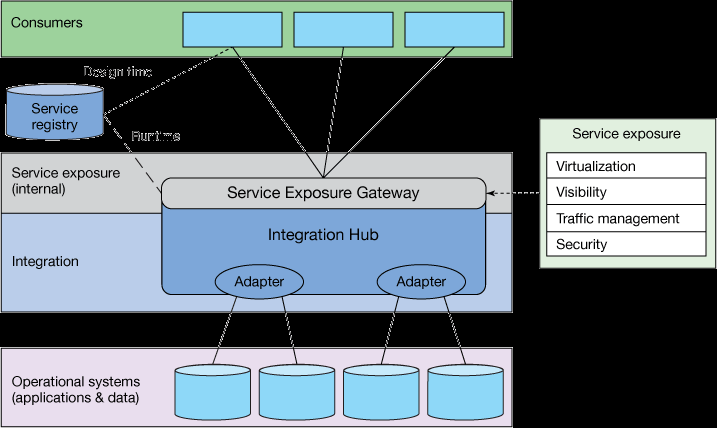
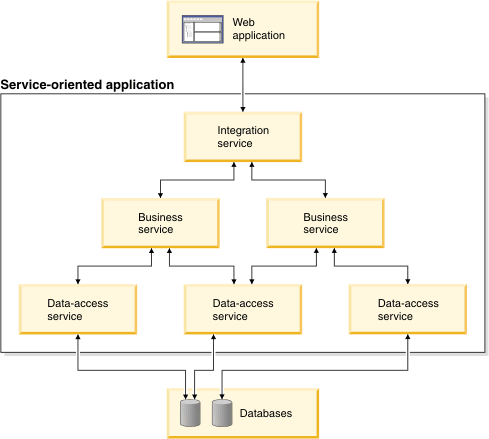
**Highly Distributed systems**



**SOA (Service-Oriented architecture)**

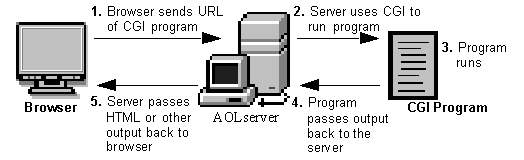
* **What is SOA?**
  + A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.
* **Why SOA?**
  + Enterprise architects believe that SOA can help businesses respond more quickly and cost-effectively to the changing market conditions. This style of architecture promotes reuse at the macro (service) level rather than micro levels (eg. objects). It can also simplify interconnection to and usage of existing IT (legacy) assets.

**SOA Architecture (Slide 39)**



**The Birth of J2EE**

**CGI**



**Problem with CGI**

* + Increase web centric application –lead problem of - resource intensive- not scalable
  + Solution :-
    - Java Servlet
    - JSP

**J2EE Architecture**

* J2EE is a layered architecture
* J2EE framework designed based on HTML, JSP, EJB, Servlets
* Using these we can design application that are Flexible, Scalable, Distributed, Component-based, multi-tier

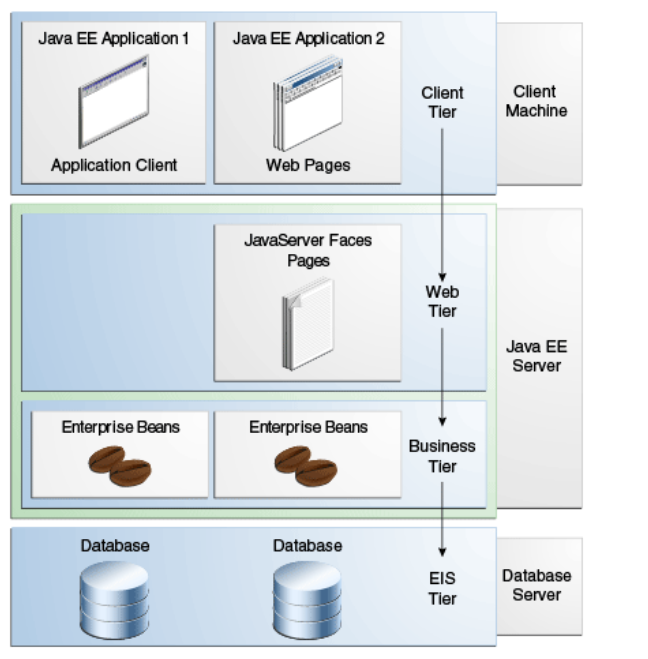
**What is J2EE (or JEE)**

* Short for ***J****ava* ***2*** *Platform* ***E****nterprise* ***E****dition*.
* J2EE is a platform-independent, Java-centric environment from Sun/Oracle for developing, building and deploying Web-based enterprise applications online.
* The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multi-tiered, Web-based applications.

**Distributed Multi-tiered Applications**

* The Java EE platform uses a distributed multi-tiered application model for enterprise applications.
* Application logic is divided into components according to function
* The application components that make up a Java EE application are installed on various machines depending on the tier in the Multi-tiered Java EE environment to which the application component belongs.

**Multi-tiered Application**



**Three-tiered Applications**

* Java EE Multi-tiered applications are generally considered to be three-tiered applications because they are distributed over three locations:
  1. Client machines
  2. The Java EE server machine, and the database or legacy machines at the back end.
  3. Three-tiered applications that run in this way extend the standard two-tiered client-and-server model by placing a multithreaded application server between the client application and back-end storage.

**Difference between Java EE and Java SE**

* Java technology is both a programming 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.
* There are several Java platforms.
* Many developers, even long-time Java programming language developers, do not understand how the different platforms relate to each other.

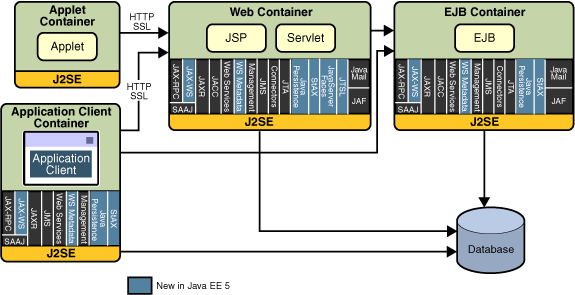
**Java SE**

* When most people think of the Java programming language, they think of the Java SE API.
* Java SE's API provides the core functionality of the Java programming language.
* It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.
* In addition to the core API, the Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java technology applications.

**Java EE**

* The Java EE platform is built on top of the Java SE platform.
* The Java EE platform provides an API and runtime environment for developing and running
  + large-scale
  + multi-tiered
  + scalable
  + reliable
  + secure network applications

**Java EE Architecture**



**Java EE Server and Container**

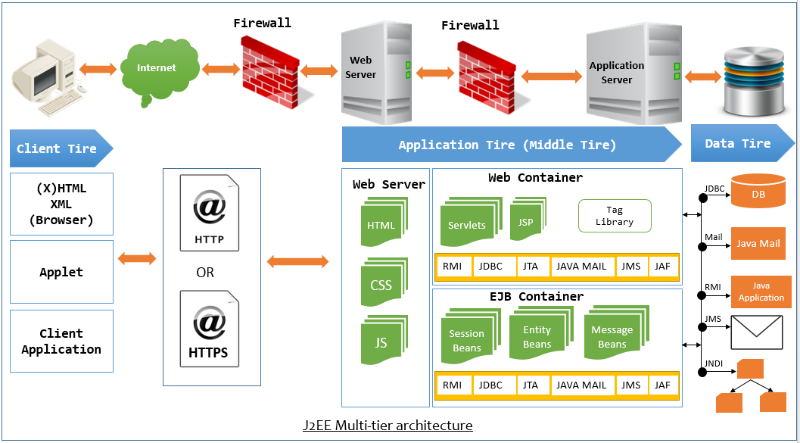
* **Java EE server**: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.
* **Enterprise JavaBeans (EJB) container**: Manages the execution of enterprise beans for Java EE applications.
* **Web container**: Manages the execution of JSP, Servlet, and Java Server Faces.
* **Application client container**: Manages the execution of application client components.
* **Applet container**: Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

**J2EE Architecture is….**

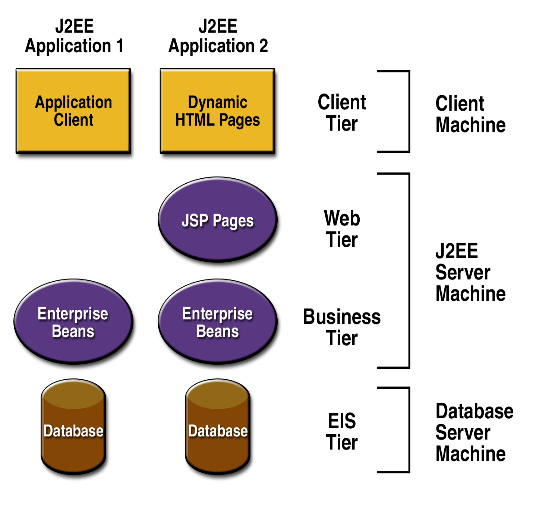
* Although four tiers but generally J2EE applications are considered to be **three tier.**
* Applications are distributed @ 3 locations as:
  + Client Machines
  + J2EE Server Machine
  + Database or legacy Machines

**J2EE is four-tier architecture**

* These consist of Client Tier (Presentation tier or Application tier), Web tier, Enterprise JavaBeans Tier (or Application server tier), and the Enterprise Information Systems Tier or the Data tier.

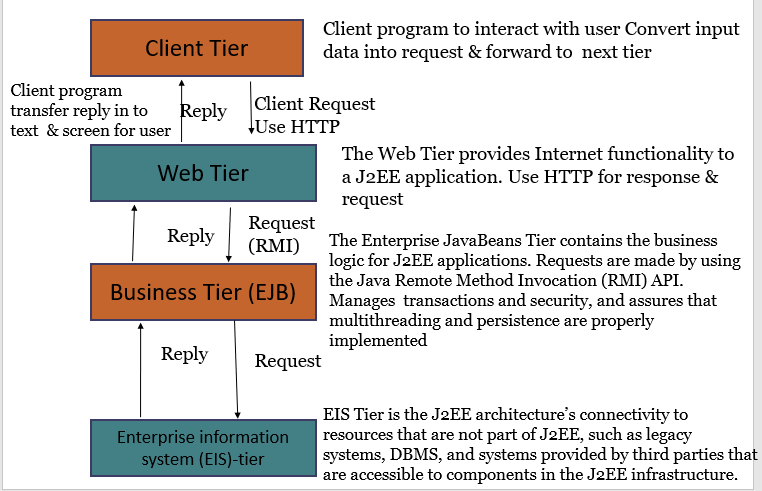


**J2EE Architecture**



* Each tier is focused on providing a specific type of functionality to an application.
* Two or more tiers can physically reside on the same Java Virtual Machine (JVM) although each tier provides a different type of functionality to a J2EE application.
* J2EE multi-tier architecture is functionally centric, a J2EE application accesses only tiers whose functionality is required by the J2EE application.

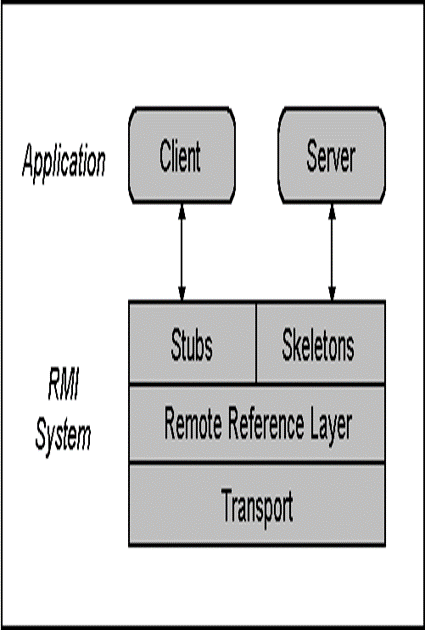
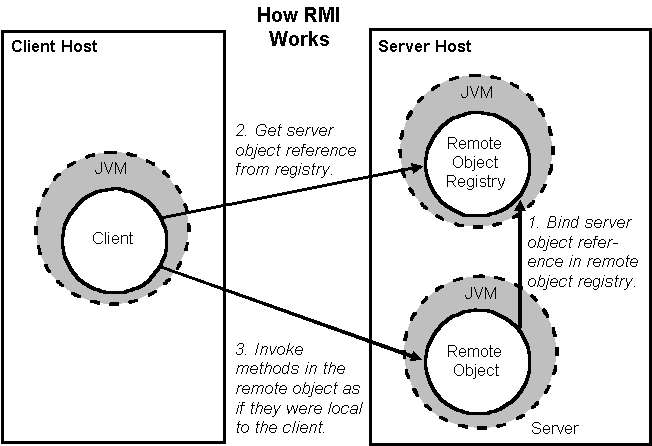
**J2EE Architecture Communication**

****

**What is RMI?**

* Java RMI (Remote Method Invocation) can be used to invoke methods of objects in another Java Virtual Machine (JVM).
* RMI allows the client (the thread that actually invokes a method in the remote object) to invoke remote methods in the server .

**RMI**



**RMI Client**

* A client invoking a method on a remote server object actually makes use of **a stub or proxy for the remote object as a channel to the remote object.**
* This stub is an implementation of the remote interfaces of the remote object and **forwards invocation requests to that server object** via the remote reference layer.
* The remote reference layer is responsible for carrying out **the semantics of the invocation**. For example the remote reference layer is **responsible for determining whether the server is a single object or is a replicated object requiring communications with multiple locations.**
* The transport is responsible for connection **set-up, connection management, and keeping track of and dispatching** to remote objects

**Skeleton**

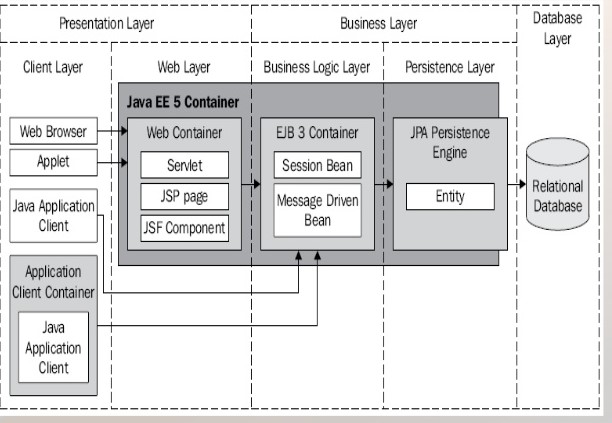
* the transport forwards the remote call up to the remote reference layer. **The remote reference layer handles any server-side behavior that needs to be done before handing off the request to the server-side skeleton. T**he skeleton for a remote object makes an up-call to the remote object implementation which carries out the actual method call.
* The return value of a call is sent back through the skeleton, remote reference layer and transport on the server side, and then up through the transport, remote reference layer and stub on the client side.

**J2EE Architecture Communication**

Access Control List (ACL):-

* Access is made using an Access Control List (ACL) that controls communication between tiers.
* The ACL is a critical design element in the J2EE multi-tier architecture because ACL bridges tiers that are typically located on different virtual local area networks.
* It adds a security level to web applications. Hackers typically focus their attack on the Web Tier to try to directly access DBMS. ACL prevents direct access to DBMS and similar resources.

**A glance at J2EE architecture**



**J2EE Clients**

**Web Clients**

* A **web client** is a software application used to locate, retrieve and display content on the World Wide **Web**, including webpages, images, video and other files. A **web client** consists of two parts:
  + dynamic web pages containing various types of markup language (HTML, XML, and so on), which are generated by web components running in the web tier, and
  + a web browser, which extracts the pages received from the server.
* A web client is sometimes called a **thin client**.
* Web clients usually do not
  + query databases,
  + execute complex business rules,
  + connect to legacy applications.

**Applets**

* An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the web browser.

**Application Clients**

* An **application client** runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language.
* It typically has a graphical user interface (GUI) created from the Swing or the Abstract Window Toolkit (AWT) API, but a command-line interface is certainly possible.
* Application clients directly access enterprise beans running in the business tier.
* **An application client can open an HTTP connection to establish communication with a servlet running in the web tier.**
* Application clients written in languages other than Java can interact with Java EE servers, enabling the Java EE platform to **interoperate with legacy systems, clients, and non-Java languages**
* **An application client** is a Java application that operates within the application client container, which is the Java 2 Runtime Environment, Standard Edition (JRE).
* An application has its own user interface and is capable of accessing all the tiers in the multi-tier architecture depending how the ACLs are configured, although typically an application has access to only the web layer.

**EJB Clients**

* Accesses one or more Enterprise JavaBeans that are located on the Enterprise JavaBeans Tier
* This access is made possible by using the RMI API.
* RMI handles communication between the Enterprise JavaBeans client and the Enterprise JavaBeans Tier using either the Java Remote Method Protocol (JRMP) or the Internet Inter-ORB Protocol (IIOP).

**EIS Clients**

* EIS clients are the interface between users and resources located on the EIS Tier.
* These clients use Java connectors, appropriate APIs, or proprietary protocols to utilize resources such as DBMS and legacy data sources.

**Web Service Peers**

* Web services provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other
* This forms a peer-to-peer relationship with other components on the Web Tier rather than a true client/server relationship.
* It uses HTTP protocol using either electronic business XML or the Simple Object Access Protocol (SOAP)
* **<message>** - The message parameter definition is used to define the different data elements for each operation performed by the web service.
* **<portType>** - This actually describes the operation which can be performed by the web service.
* **<binding>** - This element contains the protocol which is used.

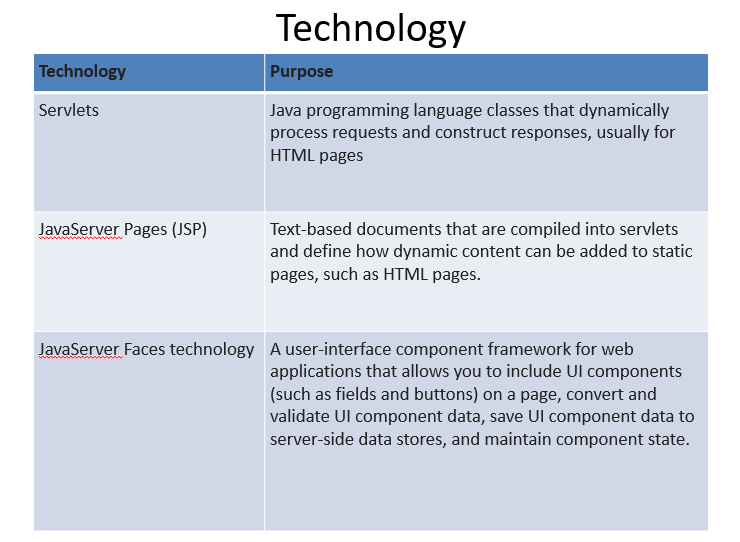
**Multi-tier Clients**

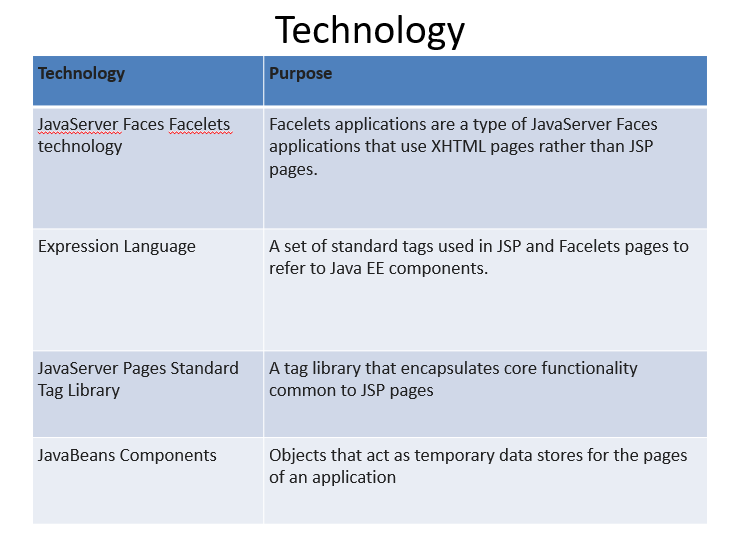
* It accesses components located on tiers other than the tier where multitier client resides.
* These clients typically use Java Message Service (JMS) to connect to other tiers.

**The Web tier**

* Performs various task:-
  + Dynamically generate content in various format of client.
  + Collect input from the client and return appropriate results from the components in the Business tire.
  + Control the flow of screens or pages on the client.
  + Maintain state of data for a user’s session
  + Perform some basic logic and hold some data temporarily in JavaBeans components.
* There are two types of components that work on the Web Tier. These are **servlets and Java Server Pages (JSP)**

**Technology - Technology**





**Servlet**

* Servlet can be described in many ways, depending on the context.
* Servlet is a technology which is used to create a web application.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page.

**Servlet Tasks**

* Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
* Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
* Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
* Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
* Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

**Servlet Architecture includes**

* **Servlet Interface**
  + Servlet interface can be implemented directly or indirectly by extending **GenericServlet** or **HttpServlet** class.
* **Request handling methods**
  + There are 3 methods defined in Servlet interface: **init(), service() and destroy()**
  + the init method :-
    - initialization .
    - It is called only once during the lifetime of a servlet. So, put all initialization code here.
  + The Service method is used for handling the client request.
    - As the client request reaches to the container it creates a thread of the servlet object, and request and response object are also created.
    - These request and response object are then passed as parameter to the service method, which then process the client request.
    - The service method in turn calls the doGet or doPost methods (if the user has extended the class from HttpServlet ).
* **Number of instances**

**Java Server Page (JSP)**

* JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
* Java Server Page (JSP) is a technology for controlling the content or appearance of Web pages through the use of servlets.

**JSP Example**

<%-- JSP comment --%>

<HTML>

<HEAD>

<TITLE>MESSAGE</TITLE>

</HEAD>

<BODY>

<%out.print("Hello, Sample JSP code");%>

</BODY>

</HTML>

**Difference between Servlet and JSP**

* **Servlets –**
  + Servlet is a Java program which supports HTML tags too.
  + Generally used for developing business layer(the complex computational code) of an enterprise application.
  + Servlets are created and maintained by Java developers.
* **JSP –**
  + JSP program is a HTML code which supports java statements too.To be more precise, JSP embed java in html using JSP tags.
  + Used for developing presentation layer of an enterprise application
  + Frequently used for designing websites and used by web developers
* The major difference between them is that servlet adds HTML code inside java while JSP adds java code inside HTML

**The Business Tire**

* Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business .
* Receives data from the client program
* Process it, if necessary
* Sends it to the Enterprise Information System Tier for storage.
* Retrieves data from storage
* Process it, if necessary
* Sends it back to the client program.

**Enterprise JavaBeans Tier**

1. This tier contains EJB server which stores and manages EJB’s
2. The Enterprise JavaBeans server and Enterprise JavaBeans container are responsible for low-level system services that are required to implement business logic of an Enterprise Java Bean.
3. These system services are- resource pooling , Distributed object protocols, Thread management, State management, Process management, Object persistence, Security, Deploy-time configuration
4. Components on the Client Tier and the Web Tier communicate with the Enterprise JavaBeans Tier using the Java RMI API and either IIOP or JRMP.

**Java EE technologies**

* Enterprise Java Beans
* Java Persistent API[JPA]:-
  + Standard API for Object Relational Mapping [ORM] with its Java Persistent Query Language.

**Enterprise Information Systems Tier**

* This Tier is the J2EE architecture’s connectivity to resources that are not part of J2EE.
  + Enterprise resource planning (ERP),
  + Mainframe transaction processing
  + Database systems,
  + other legacy information systems .
* The connectivity is made possible through the use of CORBA and Java Connectors or through proprietary protocols.

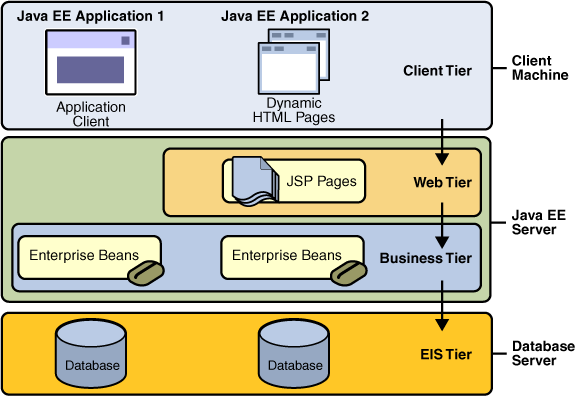
**Java EE Technology**

* JDBC
* Java Persistent API
* Java EE Connector Architecture
* Java Transaction API

**Enterprise Information Systems Tier Implementations**

* The Enterprise Information Systems (EIS) Tier is the J2EE architecture’s connectivity to resources that are not part of J2EE.
* This tier provides flexibility to developers of J2EE applications because developers can leverage existing systems and resources currently available to the corporation and do not need to replicate them in J2EE.
* EIS Tier provides the connectivity between a J2EE
* application and non-J2EE software. This connectivity is made possible through the use of CORBA and Java Connectors or through proprietary protocols.

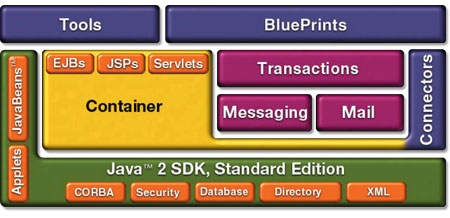
**J2EE Distributed Multitiered Application**



**J2EE Technologies**

* Java Servlets
* JSP (Java Server Pages)
* EJB (Enterprise Java Beans)
* JMS (Java Message Service)
* JDBC (Database API)
* JNDI (Java Naming & Directory Interface)
* JTA / JTS (Java Transaction API /Service)
* JavaMail
* JAAS (Java Authentication & Authorization Service)
* XML

**J2EE Components**



**J2EE Containers**

* *Containers are the interface between a component and the low-level platform-specific* functionality that supports the component.
* Before a web component, enterprise bean, or application client component can be executed, it must be assembled into a J2EE module and deployed into its container.
* J2EE support configuration of container setting for following services:-
* The J2EE security model lets you configure a web component or enterprise bean so that system resources are accessed only by authorized users.
* The J2EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
* JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access naming and directory services.
* The J2EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.
* The container also manages no configurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence.

**Container Types**

* **J2EE server:-**

The runtime portion of a J2EE product. A J2EE server provides EJB and web containers.

* **Enterprise JavaBeans (EJB) container**

Manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the J2EE server.

* **Web container**

Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

* **Application client container**

Manages the execution of application client components. Application clients and their container run on the client.

* **Applet container**

Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

**Web Services Support**

* Web services are web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients
* **XML:-** XML is a cross-platform, extensible, text-based standard for representing data. When XML data is exchanged between parties, the parties are free to create their own tags to describe the data, set up schemas to specify which tags can be used in a particular kind of XML document, and use XML stylesheets to manage the display and handling of the data.
* **SOAP Transport Protocol:-** Client requests and web service responses are transmitted as Simple Object Access Protocol (SOAP) messages over HTTP to enable a completely interoperable exchange between clients and web services, all running on different platforms and at various locations on the Internet.
* The SOAP portion of a transported message handles the following:
* Defines an XML-based envelope to describe what is in the message and how to process the message
* Includes XML-based encoding rules to express instances of application defined data types within the message
* Defines an XML-based convention for representing the request to the remote service and the resulting response

**SOAP Elements**

* + Envelope (mandatory)
    - Top element of the XML document representing the message
  + Header (optional)
    - Determines how a recipient of a SOAP message should process the message
    - Adds features to the SOAP message such as authentication, transaction management, payment, message routes, etc…
  + Body (mandatory)
    - Exchanges information intended for the recipient of the message.
    - Typical use is for RPC calls and error reporting.

**Web Services Description Language**

* WSDL is an XML language that contains information about the interface semantics and ‘administrivia’ of a call to a Web Service
* Once you develop a Web Service you publish its description and a link to it in a UDDI repository so that potential users can find it
* When someone wants to use your service, they request the WSDL file in order to find out the location of the service, the function calls and how to access them
* Then they use this information in your WSDL file to form a SOAP request to the computer
* **Definitions:**
* - WSDL is an XML-based language used to define Web Services and describe how to access them.
* - WSLD is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

**Universal Description, Discovery and Integration (UDDI)**

* Universal Description, Discovery and Integration is a SOAP application which attempts to standardize:

- How the **availability of Web (and other) services is published**

- How Web (and other) services can be (dynamically) **discovered**

* UDDI permits business to register information about themselves and their services in UDDI **Registries**
* The information held in UDDI Registries *includes:*

– **Who offers services i.e. A business/organisation**

– **What services are offered i.e. “Book Hotel Room”**

– **Where the service can be found i.e.**

* **A URL/endpoint**

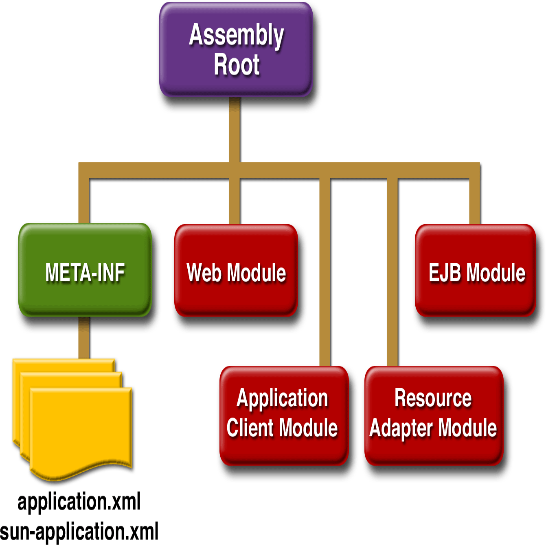
– **How the services can be accessed i.e. SOAP/CORBA**

* UDDI enables web service clients to discover potential services
* UDDI also enables businesses to discover

– Potential business partners

– Potential services to make use of

**Packaging Application**

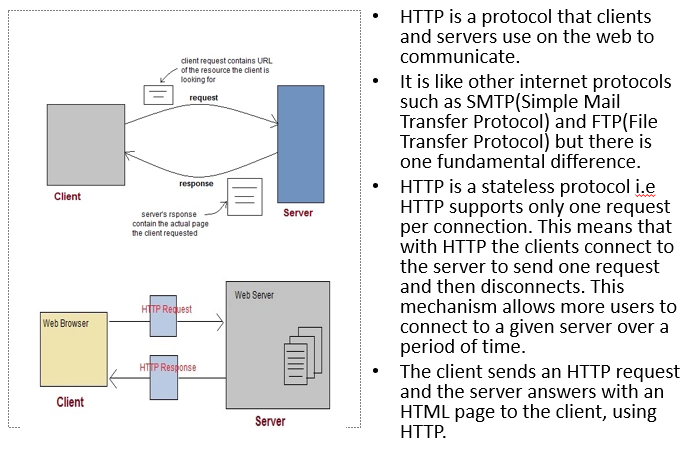
* A J2EE application is delivered in an Enterprise Archive (EAR) file, a standard Java Archive (JAR) file with an .ear extension. Using EAR files and modules makes it possible to assemble a number of different J2EE applications using some of the same components. No extra coding is needed; it is only a matter of assembling (or packaging) various J2EE modules into J2EE EAR files.
* An EAR file contains J2EE modules and deployment descriptors. A *deployment descriptor is an XML document with an .xml extension that* describes the deployment settings of an application, a module, or a component. Because deployment descriptor information is declarative, it can be changed without the need to modify the source code. At runtime, the J2EE server reads the deployment descriptor and acts upon the application, module, or component accordingly.

**NEW PPT Servlet – Lect 1**

**Prerequisite for Servlet**

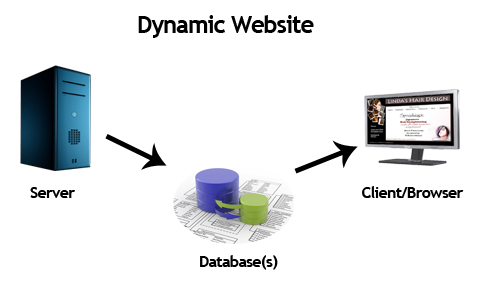
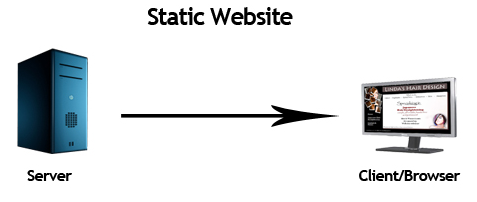
* [**https://www.youtube.com/watch?v=RsQ1tFLwldY**](https://www.youtube.com/watch?v=RsQ1tFLwldY)
* [**https://www.youtube.com/watch?v=b1HQPqTb9Ro**](https://www.youtube.com/watch?v=b1HQPqTb9Ro)
* **https://www.youtube.com/watch?v=thJSev60yfg**

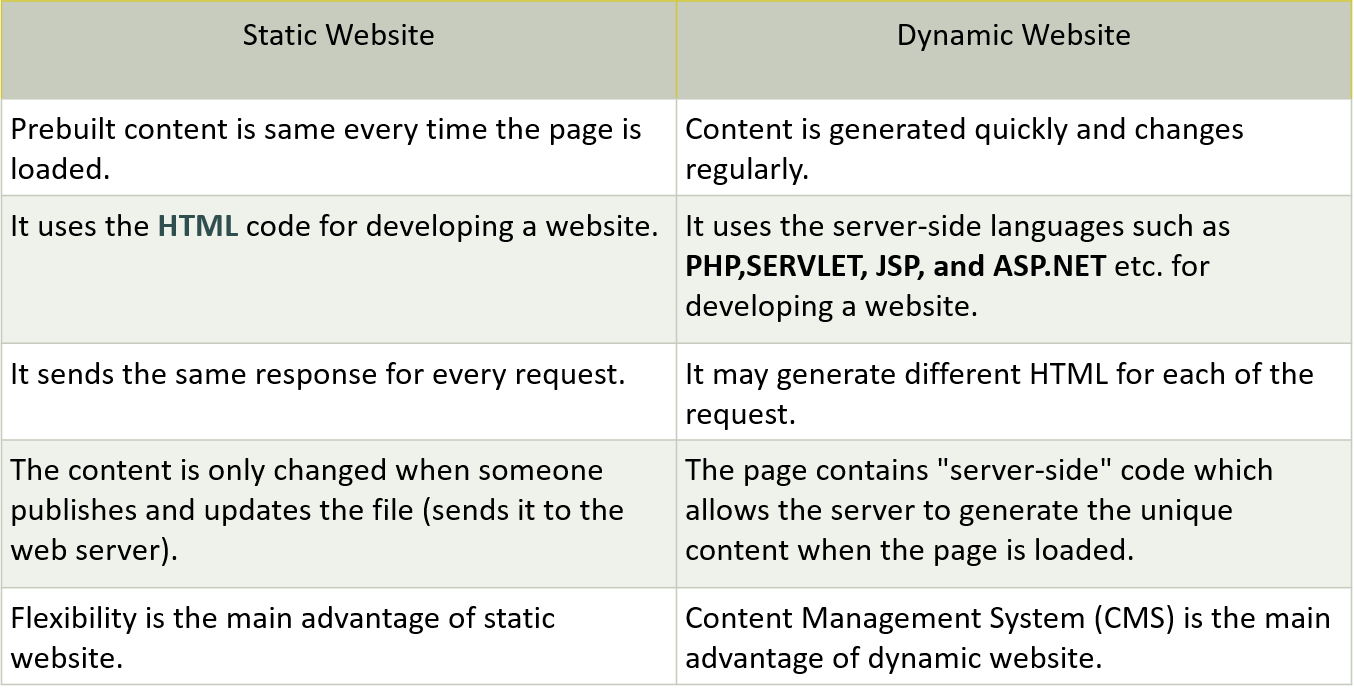
**Web Application**

****

**Difference between Static and dynamic webpage**

* The main **difference between static and dynamic web page** is that static page as name suggests remains same for all users however a dynamic web page changes based on the request from client (user’s browser).
* For example, consider a web application that shows you two input fields & an add button and when you enter two numbers and click add, it shows you another web page that has the result of addition of two numbers, this web application is dynamic in nature as the second web page that shows you the result changes based on the user input, it is not static for all users.





**WebServer**

* A web server is nothing but a software program designed to handle web requests.
* A web server is a computer system that stores, processes, and delivers web pages to clients. The client is almost always a web browser or a mobile application. Depending on the setup, a web server can store one or more websites.
* This type of server only delivers static HTML content, such as:
  + Documents
  + Images
  + Videos
  + Fonts
* It accepts incoming requests in the form of static content which is basically the components of a website including HTML pages, image and video files, etc.
* It then responds to the requests over the HTTP protocol along with optional data contents.

The main job of a web server is to serve contents to the World Wide Web to make them accessible for the end users. It may refer to a system comprised of hardware or software, or both where the web contents are stored.

**Web Server Working**



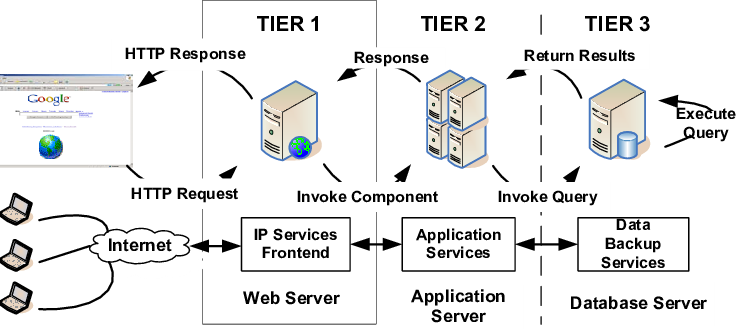
* Generating response by using the script and communicating with database.
* Sending file to the client associated with the requested URL.
* If the requested web page at the client side is not found, then web server will sends the HTTP response: Error 404 Not found.
* When the web server searching the requested page if requested page is found then it will send to the client with an HTTP response.
* If the client requests some other resources, then web server will contact to application server and data is store for constructing the HTTP response.

**Application Server**

* An application server is a program that resides on the server-side, and it’s a server programmer providing business logic behind any application
* An application server is a software framework that delivers content and assets for a client application. Clients include web-based applications, browsers, and mobile apps.
* Application servers provide clients with access to business logic. Through business logic, an app server transforms data into dynamic content and enables the functionality of the application. Examples of dynamic content are:
  + A transaction result
  + Decision support
  + Real-time analytics
  + Transaction management
  + Security
  + Dependency injection (DI)
  + Concurrency
* Application servers also handle processes such as clustering, fail-over, and load-balancing.

**Application Server Working**

* They are basically used in a web-based application that has 3 tier architecture. The position at which the application server fits in is described below:
* Tier 1 – This is a GUI interface that resides at the client end and is usually a thin client (e.g. browser)
* Tier 2 – This is called the middle tier, which consists of the Application Server.
* Tier 3 – This is the 3rd tier which is backend servers. E.g., a Database Server.



**Example of application Server**

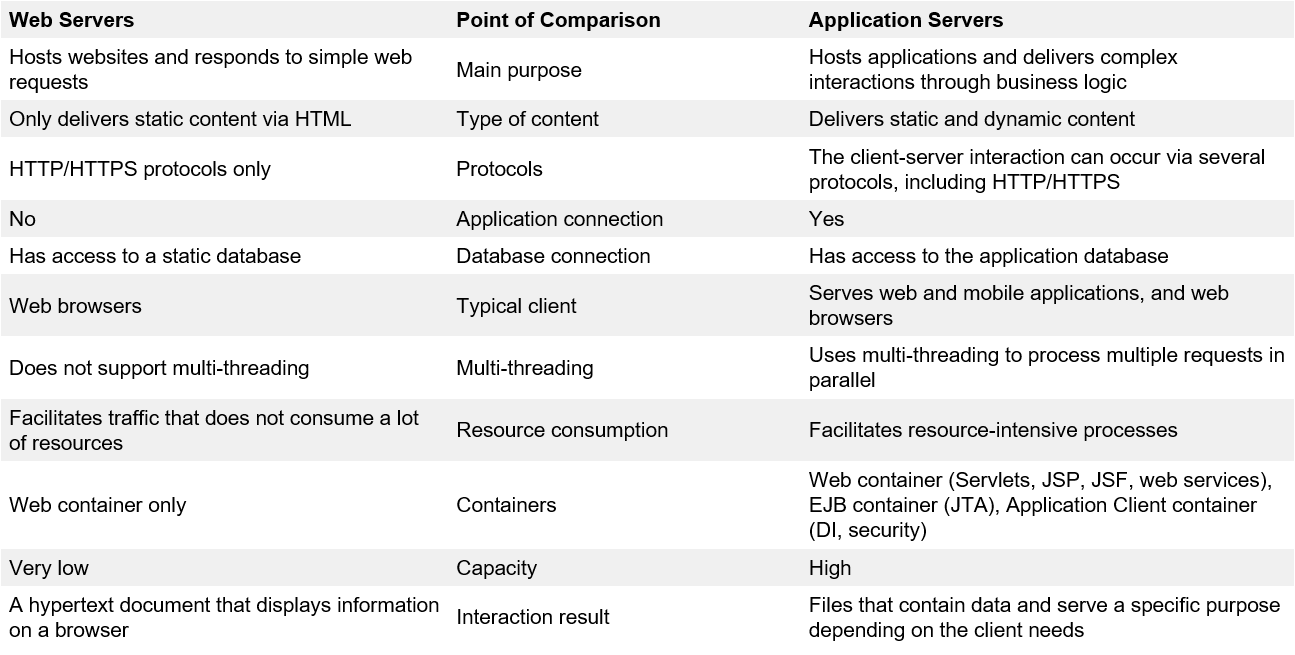
* There is a huge number of application servers that are used today. Some of the examples are given below:
* JBoss
* Weblogic
* Websphere
* Glassfish
* Tcat Server
* Apache Geronimo
* JRun
* Oracle OC4J
* Sun GlassFish Enterprise Server
* SAP Netweaver AS

Sybase Enterprise Application Server

**Webserver**

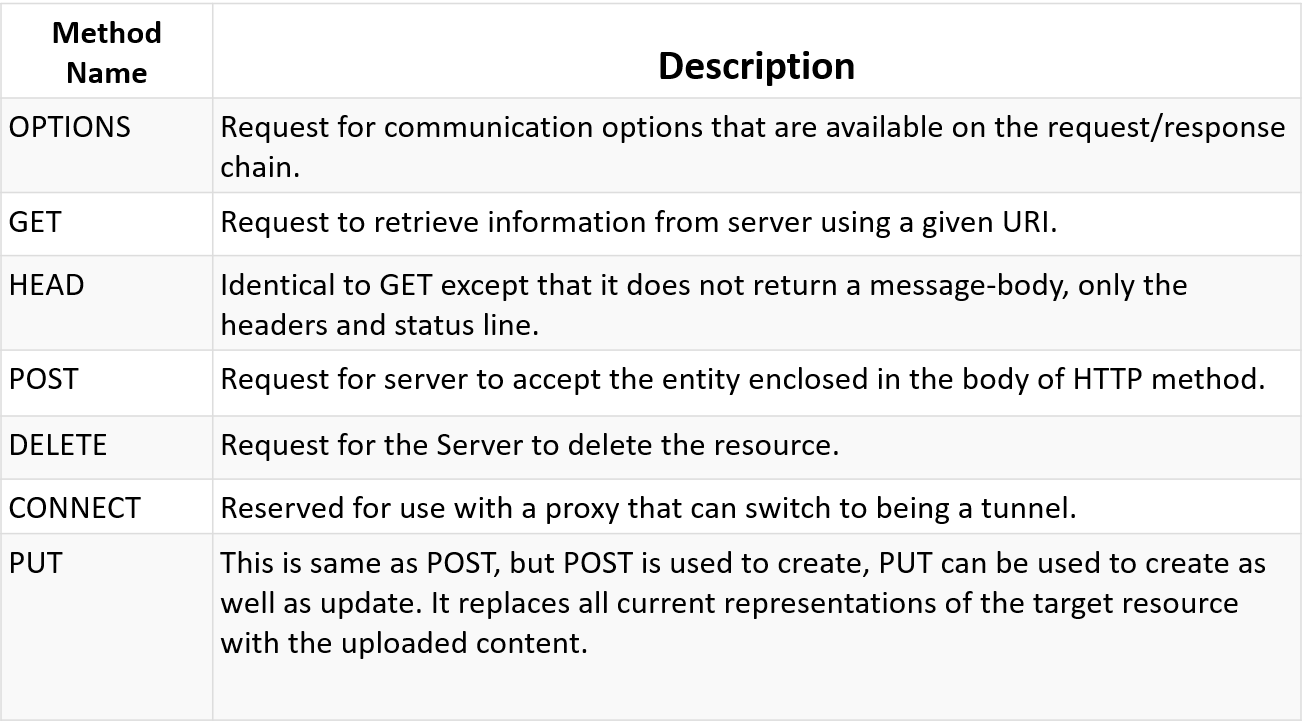
* ***Scenario 1: Web server without an application server:-***
* you have an online store with only a web server and no application server. The site will provide a display where you can choose a product from. When you submit a query, the site performs a lookup and returns an HTML result back to its client.
* The web server sends your query directly to the database server (be patient, I will explain this one in our next nugget) and waits for a response. Once received, the web server formulates the response into an HTML file and sends it to your web browser.
* This back and forth communication between the server and database server happens every time a query is run.
* ***Scenario 2: Web server with an application server***
* if the query you want to run has already been done previously and no data has changed since then, the server will generate the results without having to send the request to the database server.
* This allows a real-time query where a second client can access the same info and receive real time, reliable information without sending another duplicate query to the database server.
* The server basically acts as an intermediate between the database server and the web server. This allows the information pulled to be reusable while in the first scenario, since this info is embedded in a particular and "customized" HTML page, this is not a reusable process.
* A second client will have to request the info again and receive another HTML embedded page with the info requested -highly inefficient. Not to mention that this type of server is very flexible due to its ability to manage its own resources, including security, transaction processing, messaging and resource pooling.
* To support such a variety of complex tasks this server must have a built in redundancy, great processing power and high amount of RAM to handle all the data it's pulling in real time.

**Application server and Web server**



**HTTP Methods and Descriptions**

HTTP request can be made using a variety of methods, but the ones you will use most often are Get and Post. The method name tells the server the kind of request that is being made, and how the rest of the message will be formatted.



**Anatomy of an HTTP Get request**

* It requests the data from a specified resource
* It remains in the browser history, It can be bookmarked, can be cached, It have length restrictions, It should never be used when dealing with sensitive data, It should only be used for retrieving the data

**Anatomy of an HTTP POST request**

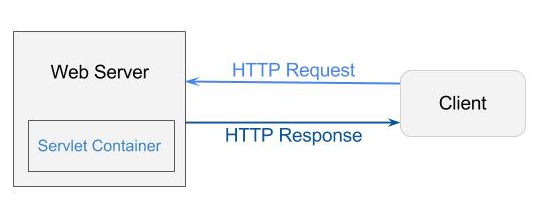
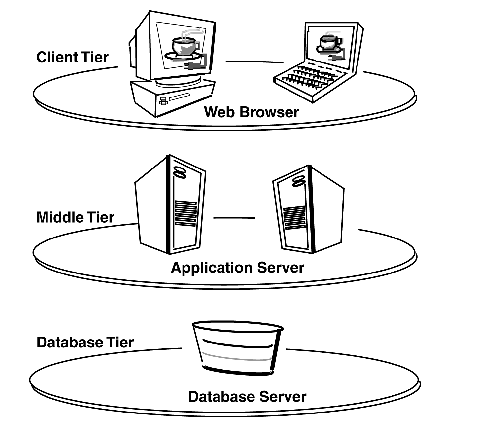
* It submits the processed data to a specified resource
* Post requests are used to make more complex requests on the server. For instance, if a user has filled a form with multiple fields and the application wants to save all the form data to the database. Then the form data will be sent to the server in POST request body, which is also known as Message body.
* This requests cannot be bookmarked ,This requests have no restrictions on length of data.
* This requests are never cached, This requests do not retain in the browser history

**Introduction to servlet (What servlets are and why you, would want to use them?**

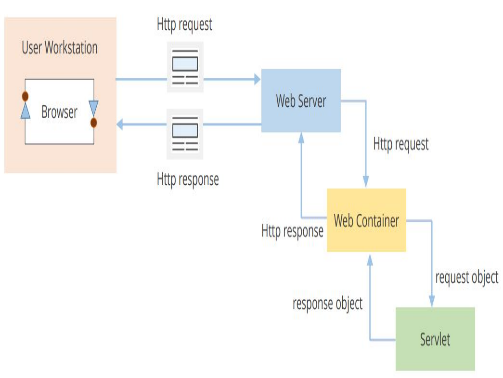
**What is servlet?**

* Servlets are server-side components. These components can be run on any platform or any server due to the core java technology which is used to implement them. Servlets augment the functionality of a web application.
* They are dynamically loaded by the server's Java runtime environment when needed. On receiving an incoming request from the client, the web server/container initiates the required servlet. The servlet processes the client request and sends the response back to the server/container, which is routed to the client.

**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.
* Using Servlets, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

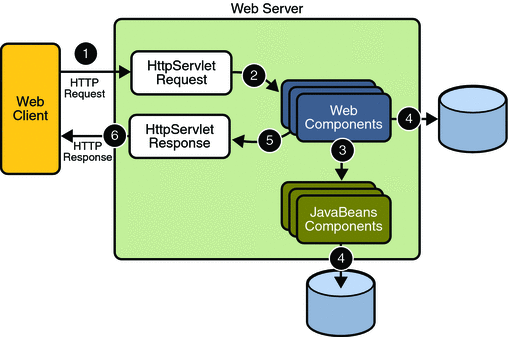
**Block diagram of Servlet Working**

* The Client browser sends an HTTP Request to the web server
* Web server forwards requests to Web Container
* Web Container forwards it to the Servlet in form of the request object
* Servlet generates response object and sends it to the Web Container
* Web container converts it into equivalent HTTP response to the web server
* The web server sends HTML content via HTTP response back to the client browser.

**How servlet works**

* **Web container** is responsible for managing execution of servlets and JSP pages for Java EE application.
* When a request comes in for a servlet, the server hands the request to the Web Container.
* **Web Container** is responsible for instantiating the servlet or creating a new thread to handle the request. Its the job of Web Container to get the request and response to the servlet.
* The container creates multiple threads to process multiple requests to a single servlet.
* **Servlets don't have a main() method**. Web Container manages the life cycle of a Servlet instance.

**What is servlet**

*  In the Java EE platform, web components provide the dynamic extension capabilities for a web server.
* Web components are either Java servlets, web pages, web service endpoints, or JSP pages.
* The client sends an HTTP request to the web server.
* A web server that implements Java Servlet and JavaServer PagesTM technology converts the request into an HTTPServletRequest object.
* This object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content.
* The web component can then generate an HTTPServletResponse or it can pass the request to another web component.
* Eventually a web component generates a HTTPServletResponse object.
* The web server converts this object to an HTTP response and returns it to the client.

**How a Servlet Application works**

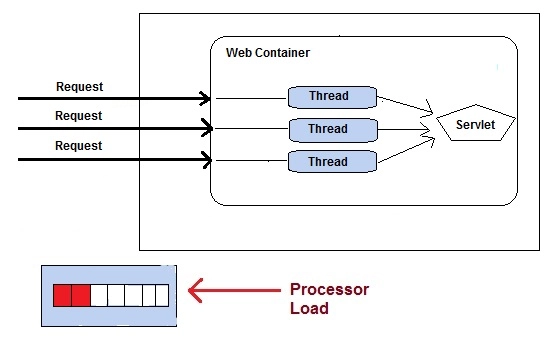
1. User sends request for a servlet by clicking a link that has URL to a servlet.
   * + **HttpServletRequest**
     + **HttpServletResponse**
2. Then the container creates or allocates a thread for that request and calls the Servlet's service() method and passes the request, response objects as arguments.
3. The service() method, then decides which servlet method, doGet() or doPost() to call, based on HTTP Request Method(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.
4. Then the Servlet uses response object to write the response.
5. After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.esponse back to the client.

**Servlet Container States**

* **Standalone:** It is typical Java-based servers in which the servlet container and the web servers are the integral part of a single program. For example:- Tomcat running by itself
* **In-process:** It is separated from the web server, because a different program runs within the address space of the main server as a plug-in. For example:- Tomcat running inside the JBoss.
* **Out-of-process:** The web server and servlet container are different programs which are run in a different process. For performing the communications between them, web server uses the plug-in provided by the servlet container.

**CGI (Common Gateway Interface)**

* **CGI technology**
  + User clicks a link that has URL to a dynamic page instead of a static page.
  + The URL decides which CGI program to execute.
  + Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
  + The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.
* **Drawback of CGI:-**
  + High response time because CGI programs execute in their own OS shell.
  + CGI is not scalable.
  + CGI programs are not always secure or object-oriented.
  + It is Platform dependent.

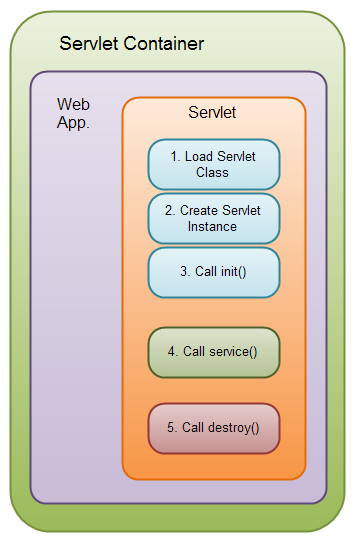
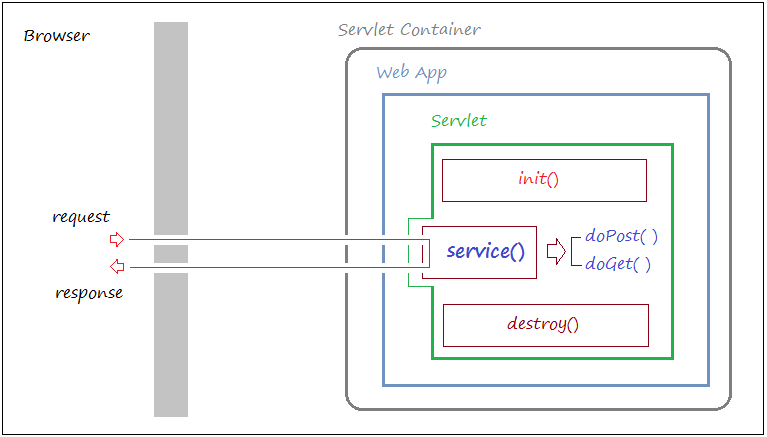
**Advantage of servlet**

Less response time because each request runs in a separate thread.

Servlets are scalable.

Servlets are robust and object oriented.

Servlets are platform independent.

**Servlet Life Cycle**

**Servlet Life Cycle**

* Loading Servlet Class :
  + A Servlet class is loaded when first request for the servlet is received by the Web Container.
  + Servlet instance creation :After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
  + Call to the init() method : init() method is called by the Web Container on servlet instance to initialize the servlet.
  + Signature of init() method :
    - **public void init(ServletConfig config) throws ServletException**
* Call to the service() method :
  + The containers call the service() method each time the request for servlet is received.
  + The service() method will then call the doGet() or doPost() methos based ont eh type of the HTTP request, as explained in previous lessons.Signature of service() method :
    - public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException
    - Call to destroy() method: The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity.

**Steps in Servlet:**

1. Load **Servlet**Class.
2. Create Instance of **Servlet**.
3. Call the servlets **init()** method.
4. Call the servlets **service()** method.
5. Call the servlets **destroy()** method.

* Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. You can force the container to load the servlet when the container starts up though.
* Step 4 is executed multiple times - once for every HTTP request to the servlet.   
  Step 5 is executed when the servlet container unloads the servlet.

**Servlet API**

* Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :
  + javax.servlet
    - package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
    - Itcontains the classes to support generic servlet (protocol-independent servlet)
  + javax.servlet.http
    - package contains interfaces and classes that are responsible for http requests only.
    - javax.servlet.http package that contains classes to support http servlet. You may be wondering what is generic and http servlet.

Syntax: java.lang.Object

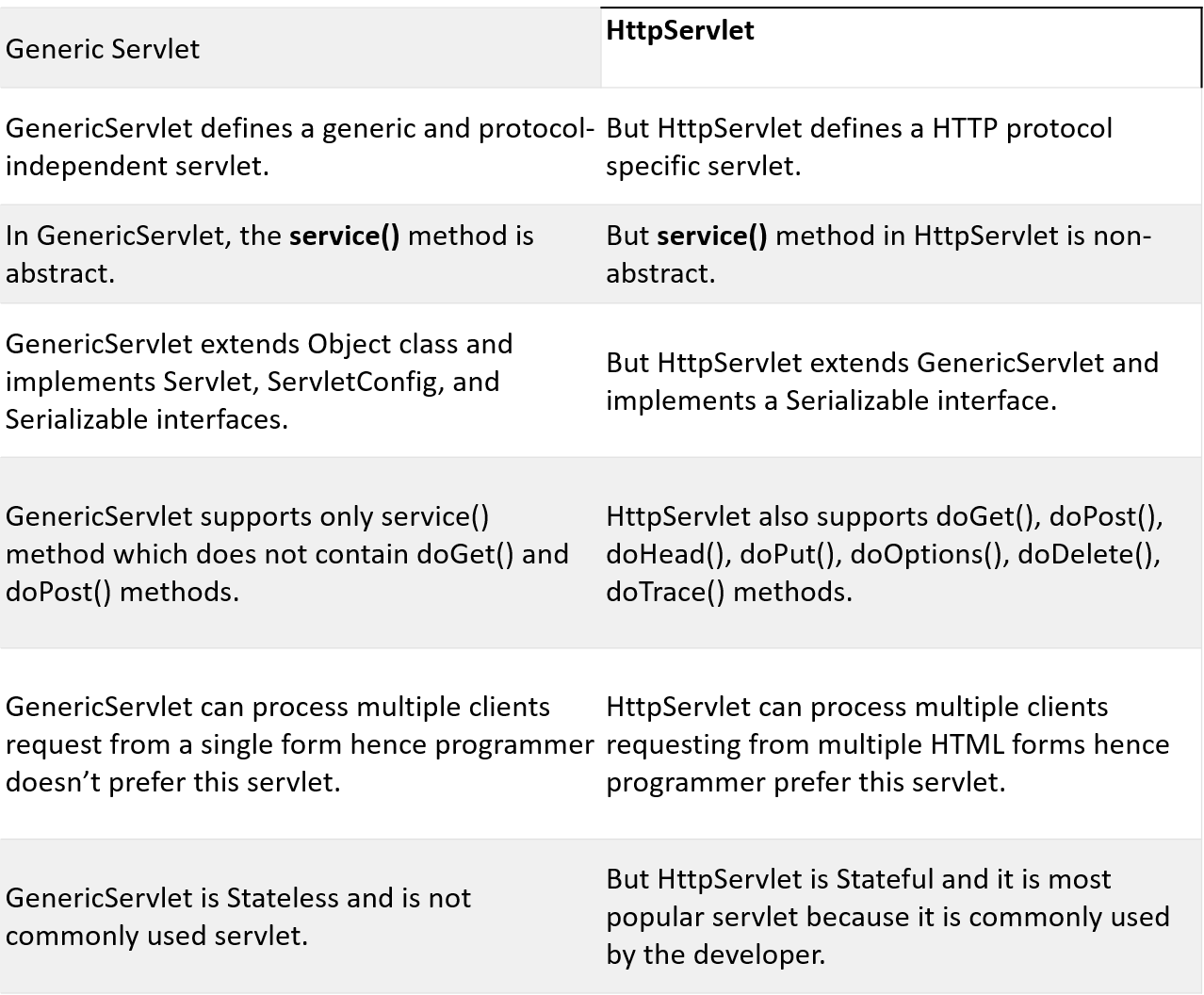
extended byjavax.servlet.GenericServlet

extended byjavax.servlet.http.HttpServlet

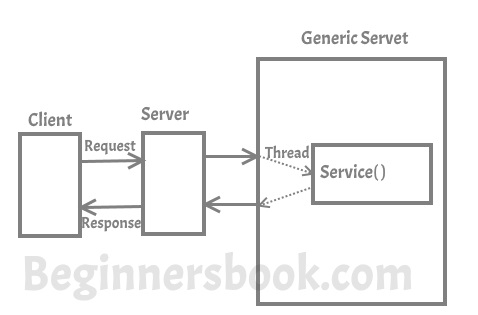
**Keywords in first Program**

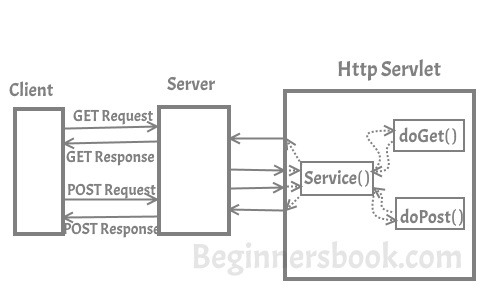
* HttpServlet
* init
* get and post
* HttpServletRequest
* HttpServletResponse
* setContentType
* response.getWriter
* PrintWtiter

**Generic and HTML Servlet**



**Generic Servlet**

* Signature of service() method:
* public abstract void service(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException
* The service() method accepts two arguments ServletRequest object and ServletResponse object. The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

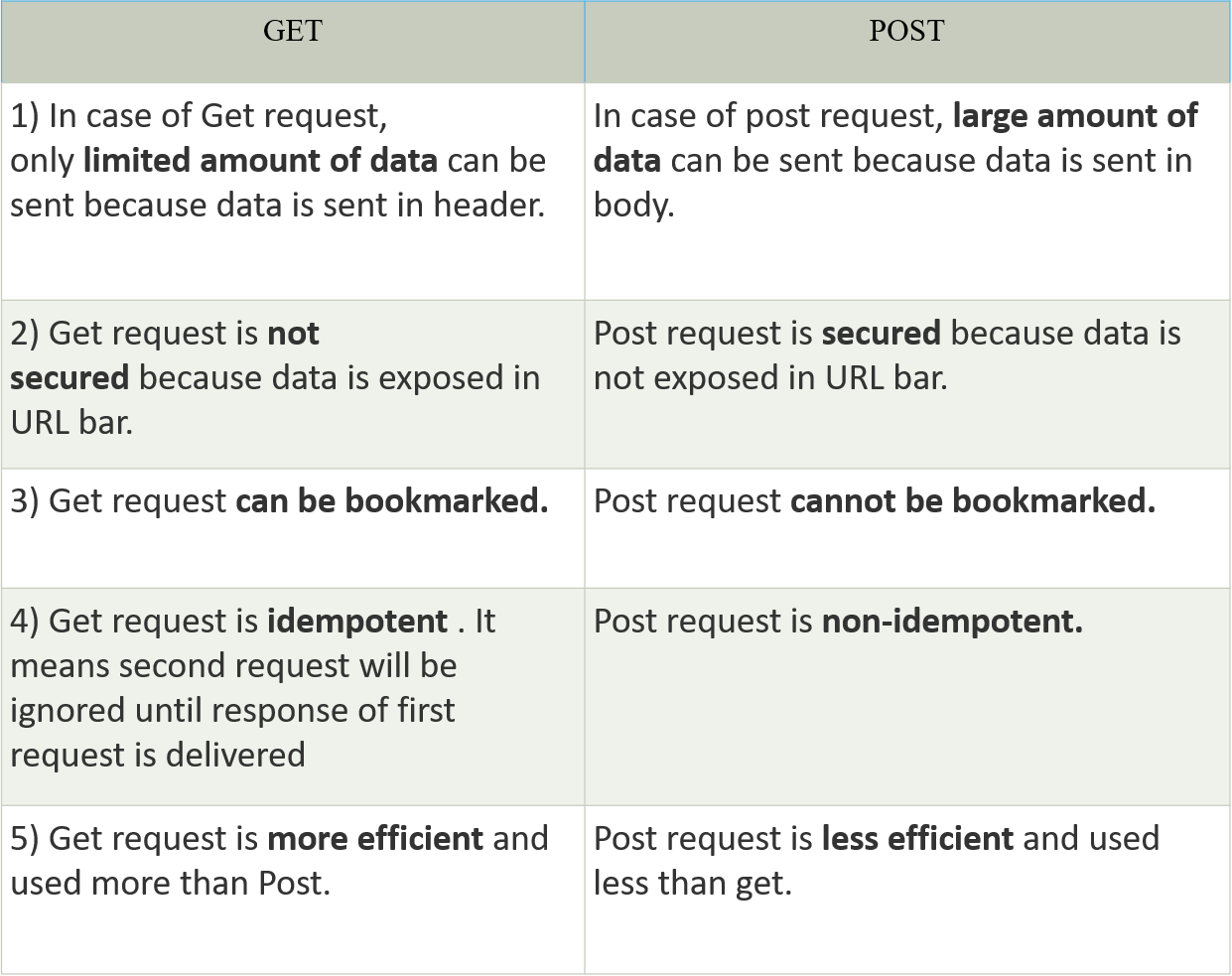
**HTTP Servlet**

* the HTTP Servlet doesn’t override the service() method. Instead it overrides one or more of the following methods. It must override at least one method from the list below:
* **doGet()** – This method is called by servlet service method to handle the HTTP GET request from client. The Get method is used for getting information from the server
* **doPost()** – Used for posting information to the Server
* **doPut()** – This method is similar to doPost method but unlike doPost method where we send information to the server, this method sends file to the server, this is similar to the FTP operation from client to server
* **doDelete()** – allows a client to delete a document, webpage or information from the server
* **init() and destroy()** – Used for managing resources that are held for the life of the servlet
* **getServletInfo()** – Returns information about the servlet, such as author, version, and copyright.

**Get and Post**

* Two common methods for the request-response between a server and client are:
  + GET- It requests the data from a specified resource
  + POST- It submits the processed data to a specified resource
  + Some other features of GET requests are:
  + It remains in the browser history
  + It can be bookmarked
  + It can be cached
  + It have length restrictions
  + It should never be used when dealing with sensitive data
  + It should only be used for retrieving the data
* Some other features of POST requests are:
  + This requests cannot be bookmarked
  + This requests have no restrictions on length of data
  + This requests are never cached
  + This requests do not retain in the browser history

**Difference between GET and POST**

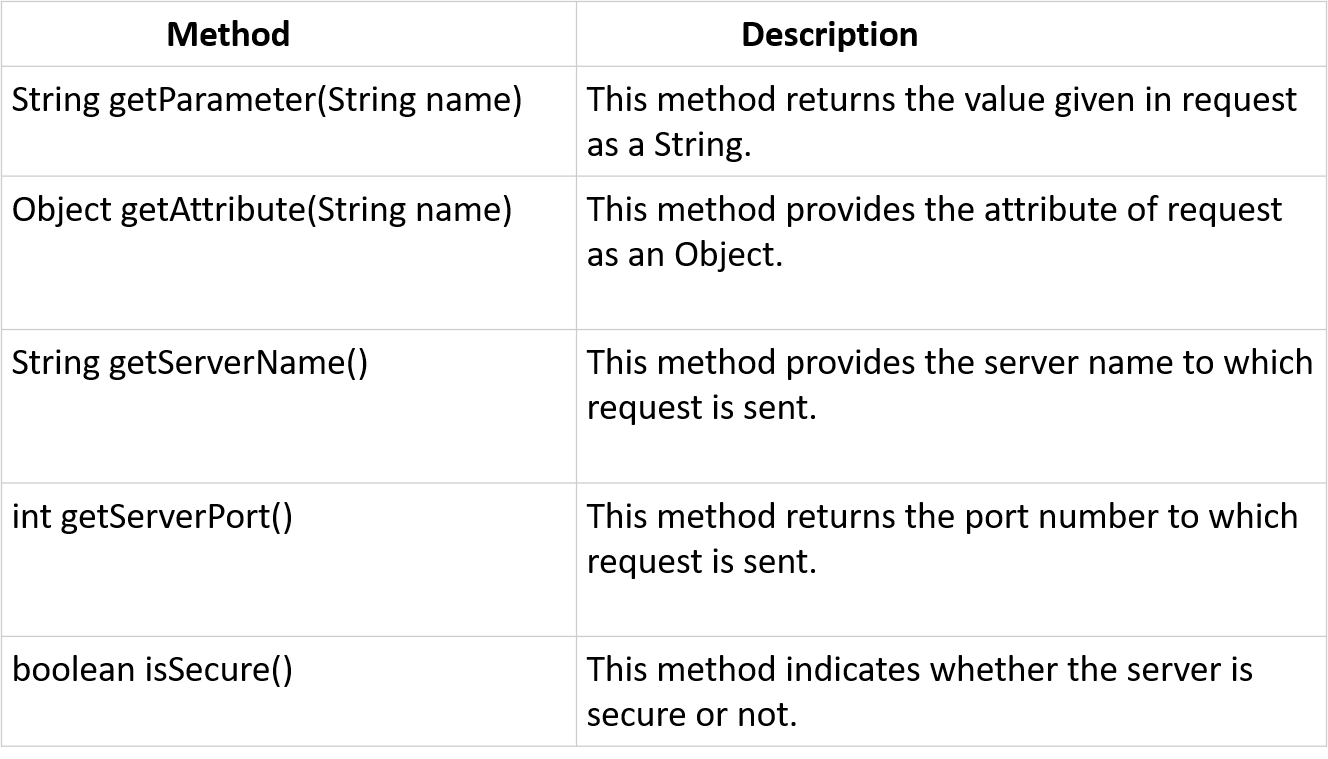


**serContentType**

* In easy words :
  + By using this method you are just specifying what kind of data you are going to send to the client.
* Technical Definition :
  + Before sending data to client , the Servlet container informs the client browser of what type of data is being sent now.
* The data that can be sent may be simple plain text, html form, xml form, image form of type gif or jpg, excel sheet etc.
* To send this information, the Servlet container uses response object with the method setContentType().
* The setContentType(String) method takes a string parameter and does not return anything (returns void).
* In "text/html", "text" is known as type and "html" known as subtype. A type contains many subtypes
* Some ways in which method is used :
  + response.setContentType("text/html"); // the most popular one
  + response.setContentType("text/plain");
  + response.setContentType("text/css"); // Cascading Style Sheet
  + response.setContentType("application/html");
  + response.setContentType("image/gif");
  + response.setContentType("application/zip");
  + response.setContentType("application/pdf");

**ServletRequest**

* ServletRequest is an interface whose object is used to provide the information of each request to servlet. This information may be any name, type, value or other attribute.

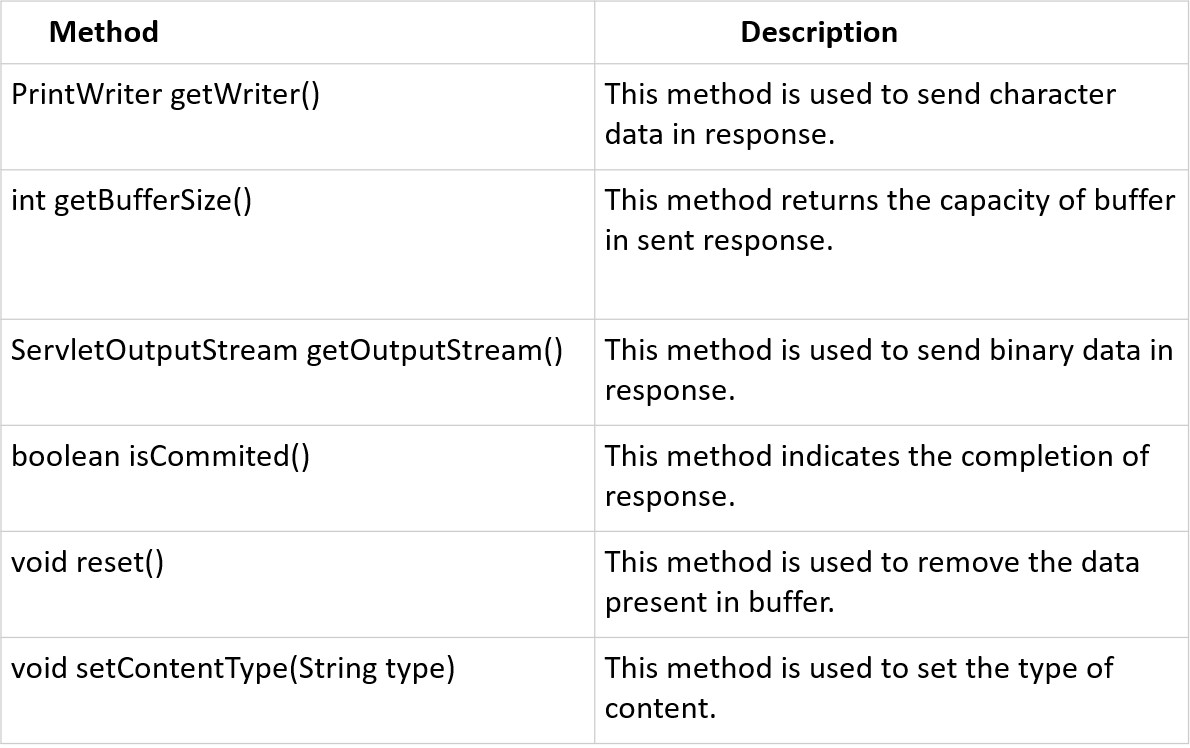


**HTTP Requests**

* The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.
* The HTTP client sends the request to the server in the form of request message which includes following information:
  + The Request-line
  + The analysis of source IP address, proxy and port
  + The analysis of destination IP address, protocol, port and host
  + The Requested URI (Uniform Resource Identifier)
  + The Request method and Content
  + The User-Agent header
  + The Connection control header
  + The Cache control header

**ServletResponse interface**

* The object of ServletResponse interface is used to send the responses to the clients. The information send in responses can be a binary or character data.
* ServletResponse interface is present in javax.servlet package and passes as an arguement of service() method.



**HTTP Response**

* HTTP Response sent by a server to the client. The response is used to provide the client with the resource it requested. It is also used to inform the client that the action requested has been carried out. It can also inform the client that an error occurred in processing its request.
* An HTTP response contains the following things:
  + Status Line
  + Response Header Fields or a series of HTTP headers
  + Message Body
* In the request message, each HTTP header is followed by a carriage returns line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used and then begins the message body.

**Status Line**

In the response message, the status line is the first line. The status line contains three items:

a**) HTTP Version Number**

It is used to show the HTTP specification to which the server has tried to make the message comply.

Example

HTTP-Version = HTTP/1.1

b) **Status Code**

It is a three-digit number that indicates the result of the request. The first digit defines the class of the response. The last two digits do not have any categorization role. There are five values for the first digit, which are as follows:

Code and Description

**1xx: Information :-**It shows that the request was received and continuing the process.

**2xx: Success:-**It shows that the action was received successfully, understood, and accepted.

**3xx: Redirection:-**It shows that further action must be taken to complete the request.

**4xx: Client Error:-**It shows that the request contains incorrect syntax, or it cannot be fulfilled.

**5xx: Server Error:-**It shows that the server failed to fulfil a valid request.

c**) Reason Phrase**

It is also known as the status text. It is a human-readable text that summarizes the meaning of the status code.

An example of the response line is as follows:

HTTP/1.1 200 OK

Here,

HTTP/1.1 is the HTTP version.

200 is the status code.

OK is the reason phrase.

**Response Header Fields**

* The HTTP Headers for the response of the server contain the information that a client can use to find out more about the response, and about the server that sent it. This information is used to assist the client with displaying the response to a user, with storing the response for the use of future, and with making further requests to the server now or in the future.

1. response-header = Accept-Ranges
2. | Age
3. | ETag
4. | Location
5. | Proxy-Authenticate
6. | Retry-After
7. | Server
8. | Vary
9. | WWW-Authenticate

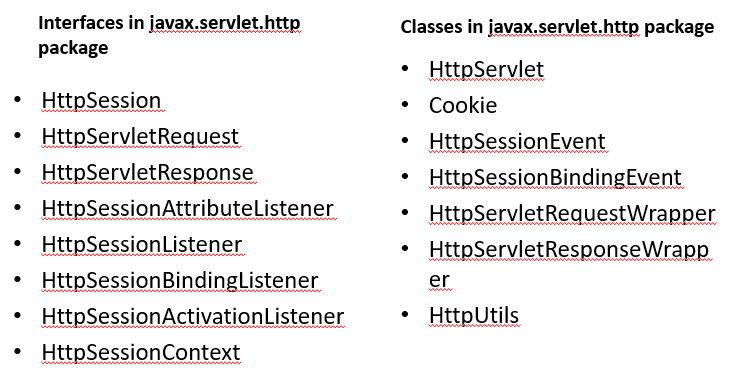
* The name of the Response-header field can be extended reliably only in combination with a change in the version of the protocol.

**Message Body**

* The response's message body may be referred to for convenience as a response body.
* The body of the message is used for most responses. The exceptions are where a server is using certain status codes and where the server is responding to a client request, which asks for the headers but not the response body.
* For a response to a successful request, the body of the message contains either some information about the status of the action which is requested by the client or the resource which is requested by the client. For the response to an unsuccessful request, the body of the message might provide further information about some action the client needs to take to complete the request successfully or about the reason for the error.

**Interfaces in javax.servlet package Classes in javax.servlet package**

* Servlet
* ServletRequest
* ServletResponse
* ServletConfig
* ServletContext
* SingleThreadModel
* RequestDispatcher
* ServletRequestListener
* ServletRequestAttributeListener
* ServletContextListener
* ServletContextAttributeListener
* Filter
* FilterConfig
* FilterChain
* GenericServlet
* ServletInputStream
* ServletOutputStream
* ServletException
* ServletRequestWrapper
* ServletRequestEvent
* ServletResponseWrapper
* ServletContextEvent
* ServletRequestAttributeEvent
* ServletContextAttributeEvent
* UnavailableException

****

**Methods of GenericServlet class**

**There are many methods in GenericServlet class. They are as follows:**

* **public void init(ServletConfig config) is used** to initialize the servlet.
* **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
* **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
* **public ServletConfig getServletConfig()** returns the object of ServletConfig.
* **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
* **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
* **public ServletContext getServletContext()** returns the object of ServletContext.
* **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
* **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
* **public String getServletName()** returns the name of the servlet object.
* **public void log(String msg)** writes the given message in the servlet log file.
* **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

**Methods of HttpServlet class**

* **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
* **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
* **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
* **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
* **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
* **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
* **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
* **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
* **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
* **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

**New Topic on Servlet**

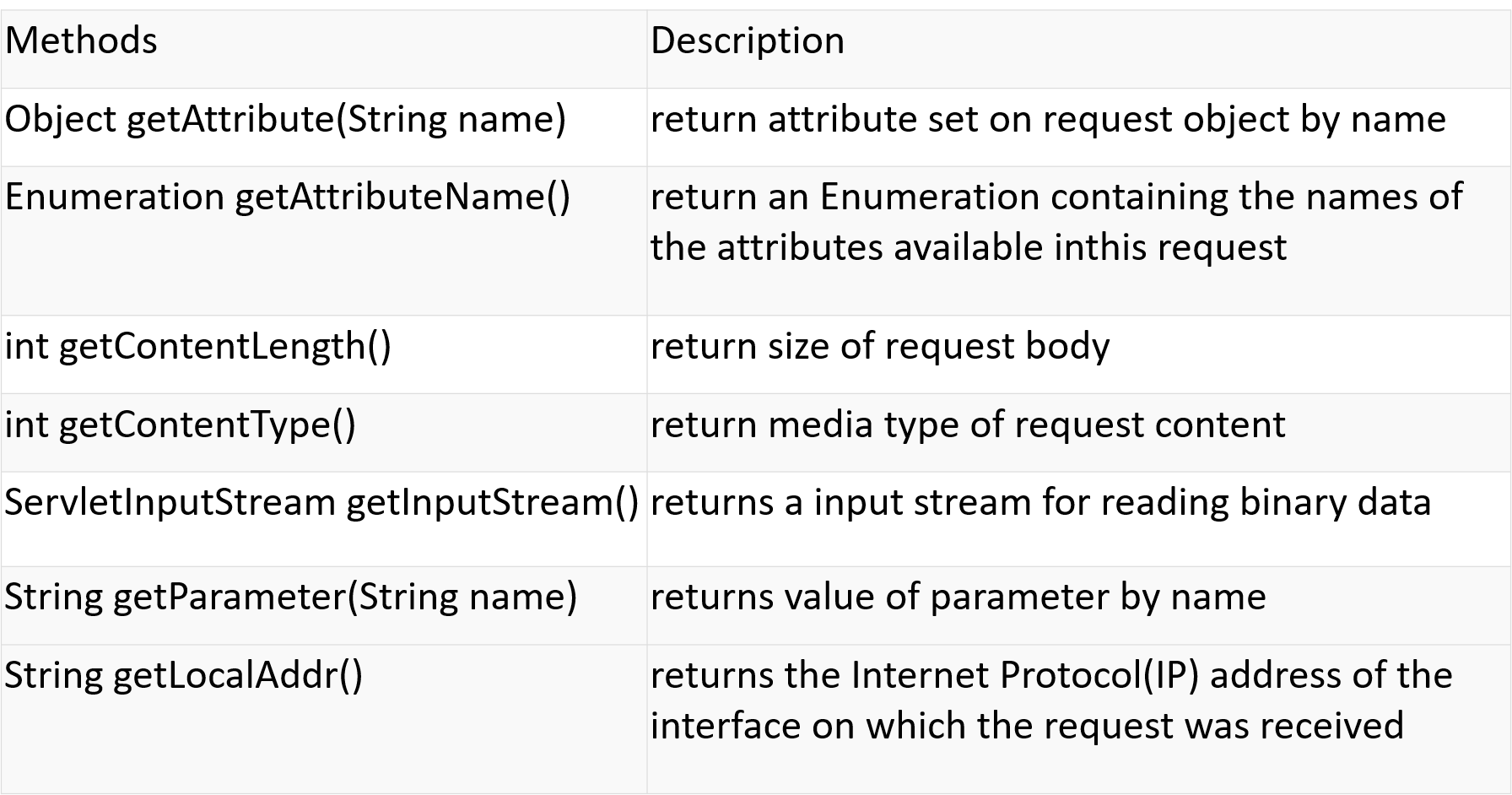
**Servlet Objects**

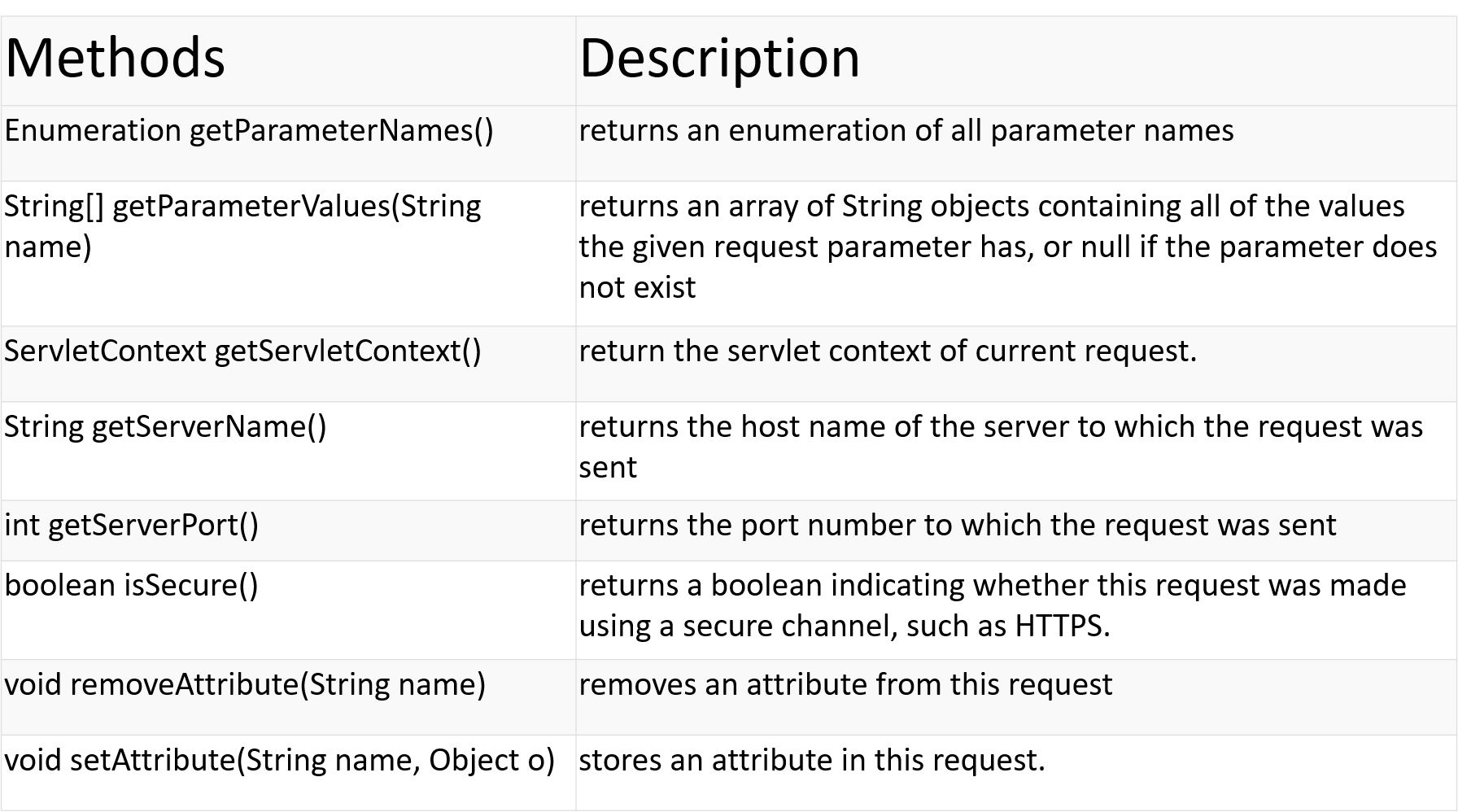
* ServletRequest:-
  + An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.
* ServletConfig:-
  + An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.
* ServletContext:-
  + An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.
* RequestDispatcher:-
  + The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
* SendRedirect:-
  + The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

**ServletRequest**

* The Servlet Request is an interface which defines different methods to handle the client requests to access a servlet, **ServletRequest** provides an instance to give the requests of client for the servlet, where servlet container establish a object and send an argument to the service method.
* Servlet API provides two important interfaces **javax.servlet.ServletRequest** and **javax.servlet.http.HttpServletRequest** to encapsulate client request. Implementation of these interfaces provide important information about client request to a servlet.

**Methods of ServletRequest interface**





**Reading Form Data**

* getParameter(java.lang.String parameterName):-
  + Take a parameter name as a parameter and returns a string object representing the corresponding value.
  + This method return null when it doesnot find a parameter of the given name.
* getParameters(java.lang.String parameterName)
  + Allow to get all parameters value for the same name parameters name return as an array of String.
  + It is used when there are multiple parameters with same name , e.g in checkbox pr combo box .
* getParameterNames()
  + Returns the parameters names in the form of an enumeration, which are used in a request.
  + This method is used with getParameter() and getParameters() methods , to obtain a list of names and values of all parameters included within request.

**Program for reading form data from the servlet**

* WAP read a name (First Name and Last Name) from the form and display
* WAP to add radio Button and read the data
* WAP check box from form

**HTTP Header – Display**

* This is set by a client to give information to a server about software that the client is used and how the client wants a server to send back the request information. The different methods are
* **getHeaderNames()**method of HttpServletRequest to read the HTTP header information. This method returns an Enumeration that contains the header information associated with the current HTTP request.
* Syntax of getHeaderNames() method
  + public Enumeration getHeaderNames()
* Syntax of getHeader() method
  + public String getHeader(String headerName)
* ***WAP- displaying all the header information of a request in the servlet page.***

**Program for display header information**

Enumeration headerNames = request.getHeaderNames();

while (headerNames.hasMoreElements()) {

String headerName = (String)headerNames.nextElement();

String headerValue = request.getHeader(headerName);

out.print("<b>"+headerName + "</b>: ");

out.println(headerValue + "<br>");

}

**HTTP Request Header**

HTTP Request Header is used to pass the additional information about the requestor itself to the server. It can be used by the client to pass the useful information. getHeaderNames() and getHeader() methods of the javax.servlet.http.HttpServletRequest interface can be used to get the header information. Following are the important header information which comes from the browser side and can be frequently used while programming:

* Accept: This specifies the certain media types that are acceptable in the response
* Accept-Charset: This indicates the character sets that are acceptable in the response. For e.g.: ISO-8859-1
* Accept-Encoding: This restricts the content-coding values that are acceptable in the response
* Accept-Language: This restricts the set of language that is preferred in the response
* Authorization: This type indicates that user agent is attempting to authenticate itself with a server
* From: This type contains the internet email address for the user who controls the requesting user agent
* Host: This type indicates the internet host and port number of the resource being requested
* If-Modified-Since: This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a 304 code (i.e. a Not Modified header) if no newer result is available
* Range: This type requests one or more sub-ranges of the entity, instead of the entire entity
* Referrer: This type enables the client to specify for the servers benefit, the address (URL) of the resources from which the request url was obtained
* User-Agent: This type contains the information about the user agent originating the request

**WAP**

* Consider the webpage containing the live score of the match or any live activities we want to see in web page in such cases we have to either refresh the web page or browse to know the current status. Servlet can help to get the solution for this problem. We can make a webpage in such a way that the webpage gets refreshed automatically after a given interval.
* We can refresh the server in two ways
  + Through client side
  + and another through server side.

**Refreshing A Servlet**

* Easiest way of refreshing the web is we can use the method setIntHeader() of the class javax.servlet.http.HttpServletResponseWrapper.
* Syntax of this method is :
  + public void setIntHeader(String headerName, int headerValue)
* For example
  + setIntHeader(“refresh”, “10”);
  + refreshes the web page every 10 second. This method sends back header “Refresh” to the browser along with an integer value. This indicates time interval in seconds.

#Coding

public class RefreshServlet extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

response.setHeader("Refresh", "3");

try (PrintWriter out = response.getWriter()) {

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet RefreshServlet</title>");

out.println("</head>");

out.println("<body>");

out.println("<h1>Servlet RefreshServlet at " + request.getContextPath() + "</h1>");

Date d = new Date();

out.println(d.toString());

out.println("</body>");

out.println("</html>");

} }

**Advantages of Refreshing Servlet**

1. Refreshing servlet may helps to know the current status of the web page For Example live cricket score.
2. Refreshing servlet help to update the current web server or in the same tab we can change the web page by assigning the url

**ServletConfig**

* An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.
* If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.
* Methods of ServletConfig interface
  1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
  2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
  3. **public String getServletName():**Returns the name of the servlet.
  4. **public ServletContext getServletContext():**Returns an object of ServletContext.

**How to get the object of ServletConfig**

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

* Syntax of getServletConfig() method

**public** ServletConfig getServletConfig();

* Example of getServletConfig() method

ServletConfig config=getServletConfig();

//Now we can call the methods of ServletConfig interface

**ServletRequest**

getInitParameter(String name)

* + This methods returns a String object that contains the value of the named initialization parameter or null, if the parameter does not exist.

getInitParameterNames()

* + This method returns the names of the initialization parameters of servlet as an enumeration of a string objects. An empty enumeration is returned by the method if servlet has no initialization parameters.

Servlet can define initial parameters by using init-param , param-name and param-value elements in the web.xml

**Init() method**

* It is called when a servlet id created for first time. It is not called for other user request.
* It is used for only one time initialization.
* Server-specific initialization parameters is done in this method
  + E.g. server might need to know various information details like database setting, password file, server specific performance parameters, hit count files or serialized cookies data from previous request.
* When we need to read the initialization parameters, we need to obtain a ServletConfig objects the getServletConfig() method, then called method getInitParameter() on the result. (i.e. this method can be used to lookup initial parameters associated with servlet .
* Setting up of the initial parameters is job of the web.xml file.

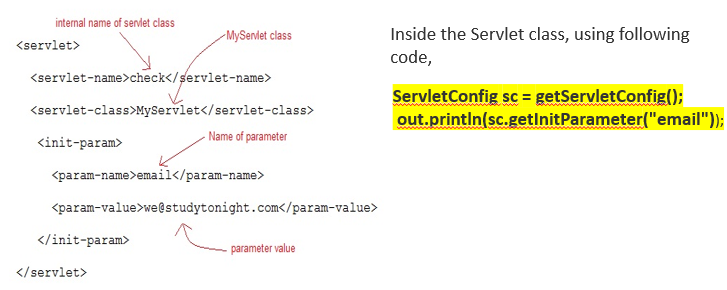
**Servlet Configuration**

* web.xml file contain initialization parameters; these parameters are required while initializing the servlet. Therefore , servlets are allowed to have initialization parameters specified outside of the compiled code and changed without required recompilation of the servlet.
* Each servlet has an object associated with it called ServletConfig. This object is created by the container and implements the javax.servlet.ServletConfig interface.
* WAP:-
  + Create a servlet in FirstApp application , which handles HTTP request and sets the value of the name attribute at the initialization of the servlet, which is displayed on the browser. The value of init-param , greeting is set in the web.xml file.

**Methods of ServletConfig**

* String getInitParameter(String name): returns a String value initialized parameter, or NULL if the parameter does not exist.
* 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.
* ServletContext getServletContext(): returns a reference to the ServletContext
* String getServletName(): returns the name of the servlet instance

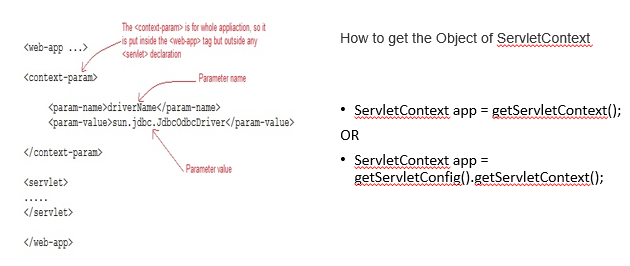
**How to inintialize a servlet inside web.xml**



**Servlet Context Object**

* An object of Servlet Context is created by the web container at time of deploying the project.
* This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.
* If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

**Servlet Context Object**

****

**Usage of Servlet Context Object**

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

**Web.xml**

<context-param>

<param-name>driverName</param-name>

<param-value>sun.jdbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet>

<servlet-name>hello</servlet-name>

<servlet-class>MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

</web-app>

**MyServlet class**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

ServletContext sc = getServletContext();

out.println(sc.getInitParameter("driverName"));

}

}

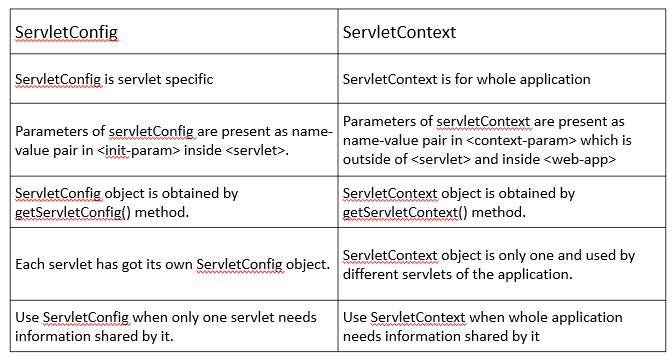
**Difference between ServletConfig and ServletContext in Java Servlet**

* **ServletConfig** and **ServletContext**, both are objects created at the time of servlet initialization and used to provide some initial parameters or configuration information to the servlet. But, the difference lies in the fact that information shared by ServletConfig is for a specific servlet, while information shared by ServletContext is available for all servlets in the web application.
* ServletConfig:
  + ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.
* ServletConfig is for a particular servlet, that means one should store servlet specific information in web.xml and retrieve them using this object.
  + Example:  
    Suppose, one is building a job portal and desires to share different email ids (which may get change over time) to recruiter and job applicant.  
    So, he decides to write two servlets one for handling recruiter’s request and another one for the job applicant.  
    Where to store email-ids?  
    Put email-id as a name-value pair for different servlet inside web.xml which can further be retrieved using getServletConfig().getInitParameter(“name”) in the servlet.

**Servlet Context**

* **ServletContext:**
  + ServletContext is the object created by Servlet Container to share initial parameters or configuration information to the whole application.
* **Example:**
  + Suppose, the name of one’s job portal is “NewWebsite.tg”. Showing the website name at the top of webpages delivered by different servlets, one needs to store the website name in every servlet inviting redundancy.
  + Since the information shared by ServletContext can be accessed by every Servlet, it is better to go with ServletContext and retrieve the website name using getServletContext.getInitParameter(“Name”) whenever required.

**Difference between ServletConfig and ServletContext**

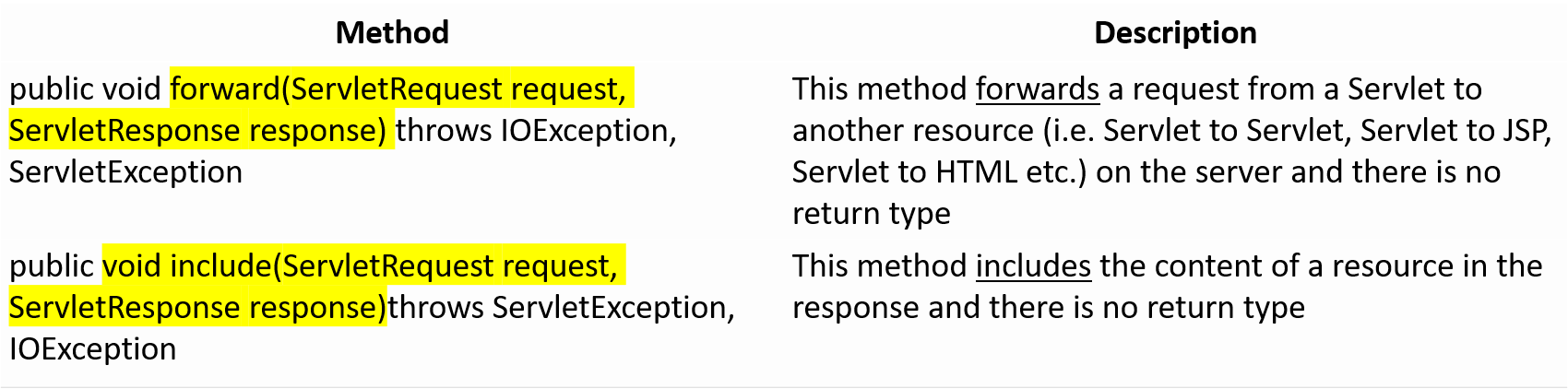


**Commonly used methods of ServletContext interface**

1. There is given some commonly used methods of ServletContext interface.**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

**RequestDispatcher in Servlet**

* While developing web applications we need to distribute the request processing and response generation to multiple servlet objects. So, we need to dispatch requests from one component to another component. This can be done by using **RequestDispatcher** interface.
* *RequestDispatcher* interface is implemented by servlet container to dispatch or to pass the request to a web resource such as Servlet, HTML page or JSP page.
* To dispatch the request from Servlet or JSP to web resource using *RequestDispatcher* we need to perform following steps:
  + Get a *RequestDispatcher* object reference
  + Using *include ()* and *forward ()* methods of *RequestDispatcher*.
* The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.
* There are two methods defined in the RequestDispatcher interface.
  + public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException
    - Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
  + public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException
    - Includes the content of a resource (servlet, JSP page, or HTML file) in the response.
* Servlet **Request Dispatcher** is an interface whose implementation defines that an object can dispatch requests to any resource (such as HTML, Image, JSP, Servlet etc.) on the server.
* Another advantage of this interface is that it is used in two cases:
  + To **include** the response of one Servlet into another (i.e. the client gets the response of both Servlets)
  + To **forward** the client request to another Servlet to honor the request (i.e. the client calls a Servlet but the response to client is given by another Servlet)
* This interface is placed in the javax.servlet package and has the following two methods:



* Both the methods are a part of Request Dispatcher interface. These methods will accept an object of the Servlet request and response interface.
* The main *difference* is that when a programmer uses forward, the control is transferred to the next Servlet or JSP the application is calling while in the case of include, the control retains with the current Servlet and it just includes the processing done by the calling of Servlet or the JSP.

**Getting a RequestDispatcher object**

**The getRequestDispacther () method of ServletContext:-**

This method takes *String* argument to locate the resource to which request is to be dispatched. When this method is called, the container locates given path. Path should start with the / character. If given path does not start with / character it throws *IllegalArgumentException*.

**The getNamedDispacther () method of ServletContext**

This method takes *String* argument used to locate Servlet to which request is to be dispatched. When this method is called, the container locates the Servlet with given name in the context.

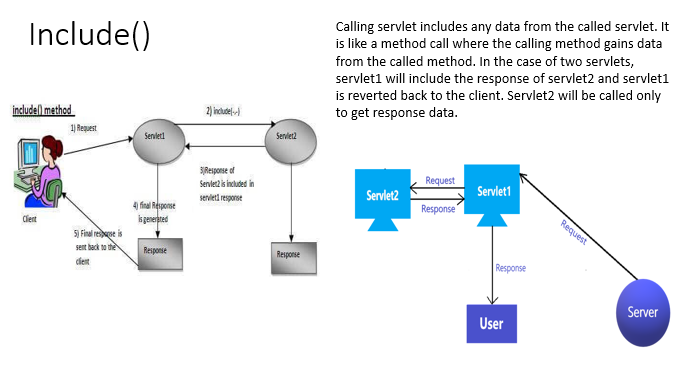
**The getRequestDispacther () method of ServletRequest**

This method obtains the *RequestDispatcher* object using path to the current request. The Servlet container builds complete path and locates the resource provided in the getRequestDispacther() method of ServletContext.

**Using include () and forward () methods of RequestDispatcher**

* **The include() method:-**
  + This method includes the response of another Servlet into the calling Servlet. This method can be invoked from calling Servlet while servicing the request. It includes contents of resource such as Servlet, JSP page or HTML page in the response. If we want generate response in the source servlet then we should make use of include () method.
* **The forward() method:-**
  + This method is used to forward requests to resource such as Servlet, JSP file or HTML page on the server. This method checks whether servlet has obtained the response and output in the response buffer. If buffer is not committed, the content is cleared. However if the buffer is already committed, it throws *IllegalStateException*. It implies that after invoking forward () method, the Servlet cannot add any response content.

**Include()**



**How to get the object of RequestDispatcher?**

* The getRequestDispatcher() method of the Servlet Request interface returns the object of the Request Dispatcher.

Syntax

RequestDispatcher rs = request.getRequestDispatcher("hello.html");

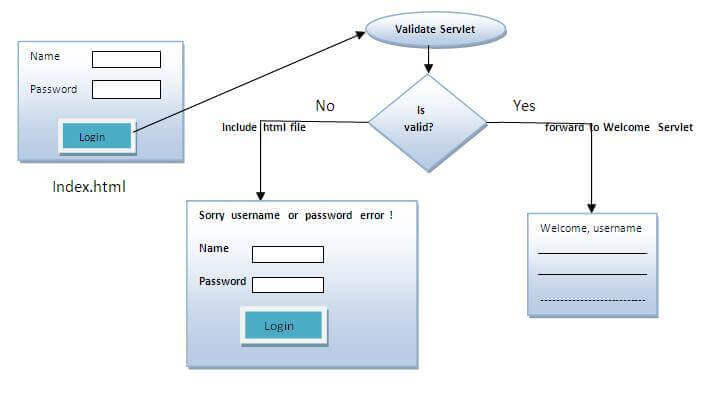
rs.forward(request,response);



**Example of RequestDispatcher interface**

* In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the Welcome Servlet, otherwise will show an error message: sorry username or password error!.
* In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:
  + **index.html file:** for getting input from the user.
  + **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
  + **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
  + **web.xml file:** a deployment descriptor file that contains the information about the servlet

**Example of RequestDispatcher interface**



* **Electrical Billing:** Electricity consumption billing.
* The bill collector takes the latest meter reading and sends to a servlet (along previous reading), say **S1** for billing.
* The **S1** servlet does not have any code to estimate the bill amount, but knows that servlet **S2** is capable. **S1** passes the client data to **S2**.
* **S2** will estimate, prepares the bill and send to **S1**.
* **S1** in turn, sends to client.
* Additional task
  + Where is the rate per unit of power consumed? Obviously, it must be with **S2** because it is estimating. No, but the rate is available with**S**1. Now **S1** has three responsibilities –
    - **a)** to send actual client data (of two readings, previous and latest) as it is to S2
    - **b)**to send rate per unit to S2 (remember this is extra information of S1 which not coming from client)
    - **c)** to receive the response of S2, add its own response, if any, like last date of bill payable etc to client. This posting does this job.

**Attribute in Servlet**

* An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

1. request scope
2. session scope
3. application scope

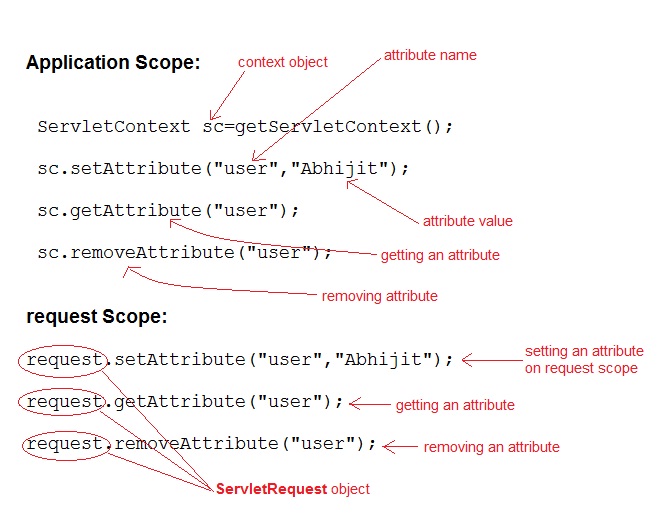
* The servlet programmer can pass information from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same objesct again and again.

**Attribute specific methods of ServletRequest, HttpSession and ServletContext interface**

There are following 4 attribute specific methods. They are as follows:

1. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
2. **public Object getAttribute(String name):**Returns the attribute for the specified name.
3. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
4. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

**Attribute in Servlet**

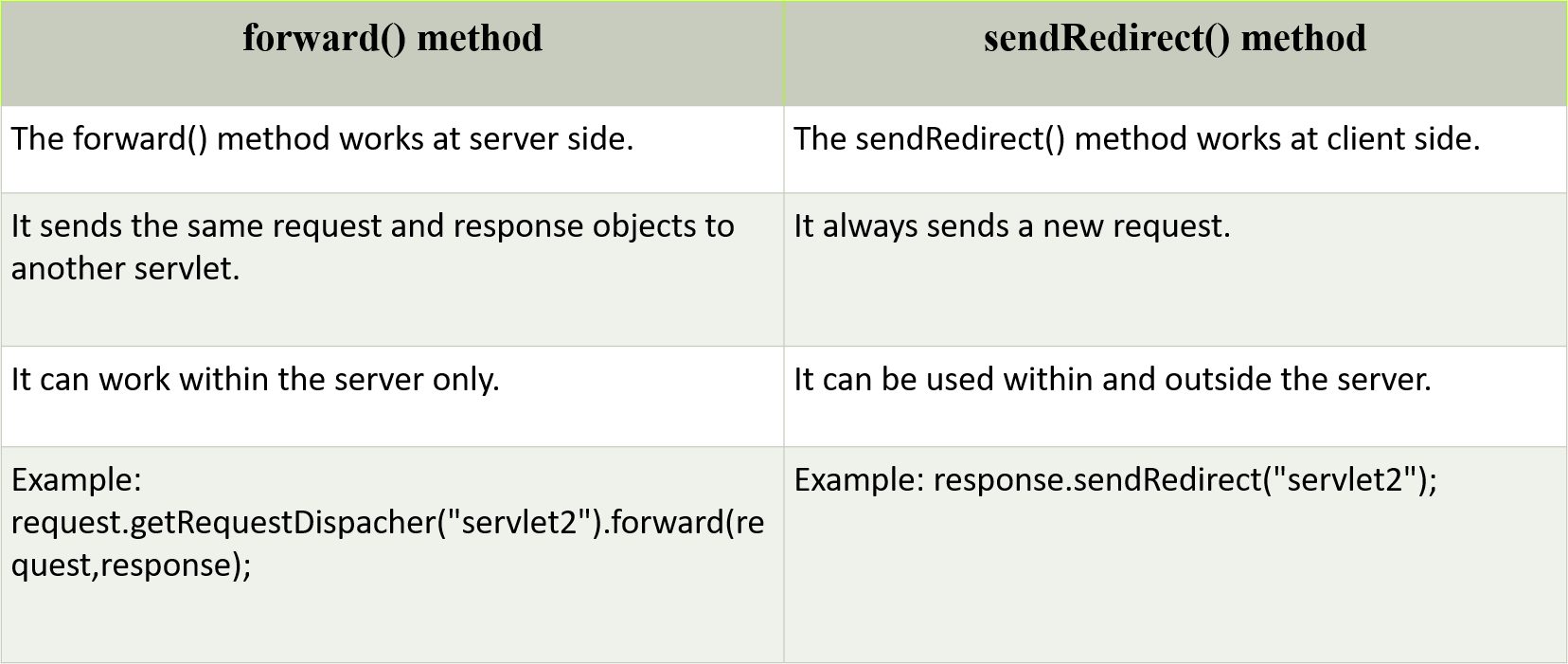


**SendRedirect in servlet**

* The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
* It accepts relative as well as absolute URL.
* It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.
* Syntax of sendRedirect() method

public void sendRedirect(String URL)throws IOException;

**Diffrence between forward() and sendRedirect() method**



**Difference between sendRedirect and RequestDispatcher**

