Semester-III

18CSP301 **J2EE** **4H – 4C**

Instruction Hours / week: L: 4 T: 0 P: 0 **Marks:** Int : **40** Ext : **60** Total: **100**

End Semester Exam : 3 Hours

Course Objectives

- To provide the necessary knowledge to design and develop database-driven application using J2EE.
- To impart expertise in Web Application Development using J2EE.

Course Outcomes(COs)

1. Understand the In-depth concepts of JEE
2. Understand the in-depth Life cycle of servlets and JSP.
3. Learn how to communicate with databases using Java.
4. Handle Errors and Exceptions in Web Applications
5. Use NetBeans IDE for creating J2EE Applications

Unit I - J2EE OVERVIEW

Beginning of Java – Java Byte code – Advantages of Java –J2EE and J2SE. J2EE Multi Tier Architecture – Distributive Systems – The Tier – Multi Tier Architecture – Client Tier Web Tier Enterprise Java Beans Tier Enterprise Information Systems Tier Implementation.

Unit II - J2EE DATABASE CONCEPTS

Data – Database – Database Schema. **JDBC Objects**: Driver Types – Packages – JDBC Process – Database Connection – Statement Objects – Result Set – Meta Data.

Unit III - JAVA SERVLETS

Benefits – Anatomy – Reading Data from Client –Reading HTTP Request Headers – Sending Data to client – Working with Cookies.

Unit IV - ENTERPRISE JAVA BEANS

Deployment Descriptors – Session Java Bean –Entity Java Bean Message Driven Bean.

Unit V – JSP

What is Java Server Pages? - Evolution of Dynamic Content Technologies – JSP & Java 2 Enterprise ed.).; **JSP Fundamentals**: Writing your first JSP- Tag conversions- Running JSP. **Programming JSP Scripts**: Scripting Languages – JSP tags- JSP directives – Scripting elements – Flow of Control – comments;

Java Remote Method Invocation.

SUGGESTED READINGS

1. Jim Keogh. (2014). The Complete Reference J2EE (1st ed.). New Delhi: Tata McGraw Hill.
   (PAGE NOS. : 3 - 61 23 - 35 98 – 116124 – 151 157 – 159 350 – 369 406 – 443 380 – 395 486- 490)
2. Duane, K. Fields., & Mark, A. Kolb. (2012). Web Development with Java Server Pages (1st ed.). Pune: Manning Publications.
   (PAGE NOS. : 2 – 15 46 - 64 65 – 99)
3. Joseph, J. Bambara et al. (2006). J2EE Unleashed (1st ed.). New Delhi:Tech Media.
4. Paul, J. Perrone., Venkata, S. R. Chaganti., Venkata S. R. Krishna., & Tom Schwenk. (2009). J2EE Developer's Handbook. New Delhi: Sams Publications.
5. Rod Johnson. (2012). J2EE Development without EJB (1st ed.). New Delhi:Wiley Dream Tech.
6. Rod Johnson., & Rod Johnson, P.H. (2012). Expert One-On-One J2ee Design and Development. New Delhi: John Wiley & Sons.

WEB SITES
1. java.sun.com/javaee/
2. java.sun.com/j2ee/1.4/docs/tutorial/doc/
3. www.j2eebrain.com/

KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore - 641021.

(For the candidates admitted from 2019 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

Faculty	Dr. V.SAMESWARI
Subject	J2EE
Subject Code	18CSP301
Class	II M.Sc Computer Science

UNIT I

SI. NO	Lecture Duration (Period)	Topics to be Covered	Support Materials
1	1	J2EE Overview: Beginning of java, Java bytecode	T1: 8-12
2	1	Advantages of java	T1: 12-15
3	1	J2EE and J2SE	T1: 15-18
4	1	J2EE Multi-tier architecture: Distributive System,	T1: 23-26, W2
5	1	The Tier	T1: 27-30
6	1	Multi-tier architecture: Client tire	T1: 30-32
7	1	Web tire	T1: 34-35
8	1	Enterprise java bean Tire	T1: 35-36
9	1	Enterprise information systems tire Implementation	T1: 36-39
10	1	Recapitulation and Discussion of Important Questions	
Total Periods Planned for Unit I			**10**

UNIT II

SI. NO	Lecture Duration (Period)	Topics to be Covered	Support Materials
1	1	J2EE Database concepts: Data, database, tables	T1: 97-99, W4
2	1	Database schema-Identifying information, Decomposing attribute to data, How to decompose attribute, example, Defining data	T1:99-107

3	1	Normalizing data& process, Grouping data,Creating Primary keys ,Functional dependency,Transitive dependencies,Foreign key,Referential Integrity	T1:108-116
4	1	JDBC OBJECTS: Driver types, packages	T1: 124-126, W4
5	1	JDBC process	T1: 126-130
6	1	Database connection	T1:130-133
7	1	Statement objects	T1:135-140
8	1	Resultset,	T1:141-144
9	1	Metadata	T1:157-160
10	1	Recapitulation and Discussion of Important Questions	
Total Periods Planned for Unit II			**10**
UNIT III			
SI. NO	**Lecture Duration (Period)**	**Topics to be Covered**	**Support Materials**
1	1	Java Servlets: Benefits	T1:350-352 W3
2	1	Anatomy of a java servlet	T1:352-353
3	1	Reading data from a client	T1:354-355
4	1	Reading HTTP request headers	T1:355-358
5	1	Sending data to a client	T1:359-361
6	1	Writing the HTTP response header	T1:362-364
7	1	Working with cookies	T1:364--367
8	1	Recapitulation and Discussion of Important Questions	
Total Periods Planned for Unit III			**8**
UNIT IV			
SI. NO	**Lecture Duration (Period)**	**Topics to be Covered**	**Support Materials**
1	1	Enterprise java bean:Introduction	T1:405-407
2	1	Deployment descriptors-Anatomy of deployment descriptor	T1:409-411
3	1	Environment Elements,Referencing EJB, Reference other resources	T1:411-419
4	1	Sharing resources,Security elements	T1:420-423
5	1	Query elements and relationship elements,	T1:423-424

6	1	Assembly elements	T1:424-431
7	1	and Exclude list elements	T1:431-434
8	1	Entity java bean	T1:434-440
9	1	Message driven bean	T1:441-444
10	1	Recapitulation and Discussion of Important Questions	
Total Periods Planned for Unit IV			**10**

UNIT V

SI. NO	**Lecture Duration (Period)**	**Topics to be Covered**	**Support Materials**
1	1	JSP: What is java server page, evolution of dynamic content technologies	T2: 1-9
2	1	JSP & Java 2 Enterprise Edition	T2:10-12
3	1	JSP Fundamentals- Writing your first JSP , Tag conversion, Running JSP	T2:21-40
4	1	Programming JSP Scripts: Scripting languages, JSP tags, JSP directives	T2:50-65
5	1	Scripting elements, flow of control, comments	T2:67-80
6	1	Java remote method invocation	W3
7	1	Recapitulation and Discussion of Important Questions	
8	1	Discussion of Previous ESE Question Papers	
9	1	Discussion of Previous ESE Question Papers	
10	1	Discussion of Previous ESE Question Papers	
Total Periods Planned for Unit V			**10**
Total Periods			**48**

TEXT BOOKS	
T1	Jim Keogh, (2002), "The Complete Reference J2EE", 1st Edition, Tata Mcgraw Hill Publications, New Delhi.
T2	Duane K fields And Mark A. Koib (2000), "Web Development with Java Server Pages", 1st Edition, Manning Publications, Pune.
References	
R1	Joseph. J. Bambara et. Al. (2001), "J2EE Unleashed", 1st Edition, New Delhi.

R2	Paul. J. Perrone, Venkata. S. R. Chaganti, Venkata. S.R.Krishna and Tom Schwenk, (2003), "J2EE Developers", Havel Book, Sams Publications, New Delhi.
R3	Rod Jhonson, (2004), "J2EE Development without EJB", Wiley Dream Tech, New Delhi.
R4	Rod Jhonson, (2004), "Expert One-on-one J2EE Design and Development", Jhon Wiley and sons, New Delhi.
Web Sites	
W1	java.sun.com
W2	www.java-Forums.org.
W3	http://www.slideshare.net/Aamir97/j2ee-multitier-architecture
W4	www.javatpoint.com
W5	https://docs.oracle.com

UNIT I

J2EE OVERVIEW

J2EE is Java, optimized for enterprise computing. Officially J2EE stands for Java 2 Platform, Enterprise Edition. J2EE is an open, standard-based, development and deployment platform for building n-tier, web-based and server-centric and component-based enterprise applications. As an enterprise platform, the J2EE environment extends basic Java with tools that "provide a complete, stable, secure, and fast Java platform to the enterprise level." One goal of using J2EE is reducing the cost and complexity of creating large-scale solutions. Because Java is a strongly typed language, use of the language is often inherently more secure in Web applications than Web applications built with less strong typing

BEGINNING OF JAVA

Java was created in 1991. It was developed by James Gosling et al. of Sun Microsystems. Initially called Oak, in honor of the tree outside Gosling's window, its name was changed to Java because there was already a language called Oak. The original motivation for Java is the need for platform independent language that could be embedded in various consumer electronic products like toasters and refrigerators. As a programming language, Java can create all kinds of applications that you could create using any conventional programming language

JAVA BYTE CODE

Java bytecode is the form of instructions that the Java virtual machine executes. Each bytecode opcode is one byte in length, although some require parameters, resulting in some multi-byte instructions. Not all of the possible 256 opcodes are used. Java bytecode is designed to be executed in a Java virtual machine. There are several virtual machines available today, both free and commercial products. Fig.1.1 shows the process of converting a souce code to byte code.

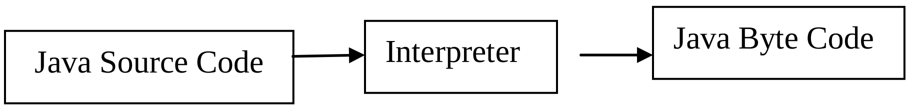


Fig. 1.1 Converting Source code to bytecode

ADVANTAGES OF JAVA

JAVA offers a number of advantages to developers.

- **Java is simple**: Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages. The reason that why Java is much simpler than C++ is because Java uses automatic memory allocation and garbage collection where else C++ requires the programmer to allocate memory and to collect garbage.

- **Java is object-oriented**: Java is object-oriented because programming in Java is centered on creating objects, manipulating objects, and making objects work together. This allows you to create modular programs and reusable code.
- **Java is platform-independent**: One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.
- **Java is distributed**: Distributed computing involves several computers on a network working together. Java is designed to make distributed computing easy with the networking capability that is inherently integrated into it. Writing network programs in Java is like sending and receiving data to and from a file. For example, the diagram below shows three programs running on three different systems, communicating with each other to perform a joint task.
- **Java is interpreted**: An interpreter is needed in order to run Java programs. The programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter. With Java, the program need only be compiled once, and the bytecode generated by the Java compiler can run on any platform.
- **Java is secure**: Java is one of the first programming languages to consider security as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind.
- **Java is robust**: Robust means reliable and no programming language can really assure reliability. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages.
- **Java is multithreaded**: Multithreaded is the capability for a program to perform several tasks simultaneously within a program. In Java, multithreaded programming has been smoothly integrated into it, while in other languages, operating system-specific procedures have to be called in order to enable multithreading. Multithreading is a necessity in visual and network programming

J2EE AND J2SE

J2SE is considered the foundation edition of the Java platform and programming environment in which all other editions are based. J2EE is the edition of the Java 2 platform targeted at developing multi-tier enterprise applications.J2EE consists of a set of specifications, APIs and technologies defining enterprise application development. J2EE technology providers expose tools, frameworks and platforms that handle a good deal of the details of enterprise application infrastructure and behavior. J2EE implementations enjoy all of the features of the Java 2 Standard Edition (J2SE) platform with additional frameworks and libraries added to support distributed/Web development

J2EE MULTI TIER ARCHITECTURE

The J2EE platform uses a multitiered distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application are installed on different machines depending on the tier in the multitiered J2EE environment to which the application component belongs. Figure shows two multitiered J2EE applications divided into the tiers described in the following list.

- Client-tier components run on the client machine.
- Web-tier components run on the J2EE server
- Enterprise JavaBean tier components run on the J2EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

Although a J2EE application can consist of the three or four tiers shown in Figure 1.2, J2EE multitiered applications are generally considered to be three tiered applications because they are distributed over three different locations: client machines, the J2EE server machine, and the database or legacy machines at the back end. 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.

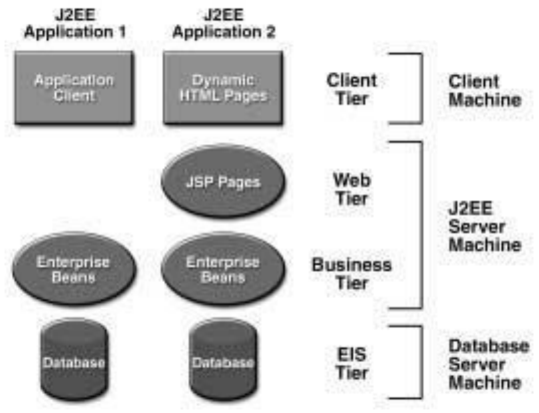


Figure1.2 J2EE Multitiered Applications

DISTRIBUTIVE SYSTEMS

The concept of multi-tier architecture has evolved over decades, following a similar evolutionary course as programming languages. The key objective of multi-tier architecture is to share resources amongst clients, which are the fundamental design philosophy used to develop programs. In earlier days programmers originally used assembly language to create programs. These programs employed the concept of software services that were shared with the program running on the machine. Software services consist of subroutines written in assembly language that communicate with each other using machine registers, which are memory spaces within the CPU of a machine. Whenever a programmer required functionality provided by a software service, the programmer called the appropriate assembly language subroutine from within the program.

Although the technique of using software services made creating programs efficient by reusing code, there was a drawback. Assembly language subroutines were machine specific and couldn’t be easily replicated on different machines. This meant that subroutines had to be rewritten for each machine. The introduction of FORTRAN and COBOL brought the next evolution of programming languages and with it the next evolution of software services. Programs written in FORTRAN could share functionality by using functions instead of assembly language subroutines. The same was true of programs written in COBOL. A function is conceptually similar to a Java method, which is a group of statements that perform a specific functionality. The group is named, and is callable from within a program. Although both assembly language subroutines and functions are executed in a single memory space, functions had a critical advantage over assembly language subroutines.

A function could run on different machines by recompiling the function. However, software services were restricted to a machine. This meant programs and functions that comprise software services had to reside on the same machine. A program couldn’t call a software service that was contained on a different machine. Programs and software services were saddled with the same limitations that affected data exchange at that time. Magnetic tapes were used to transfer data, programs, and software services to another machine. There wasn’t a real-time transmission system.

The Tier

A tier is an abstract concept that defines a group of technologies that provide one or more services to its clients. A good way to understand a tier structure’s organization is to draw a parallel to a typical large corporation (see Figure 1.3).

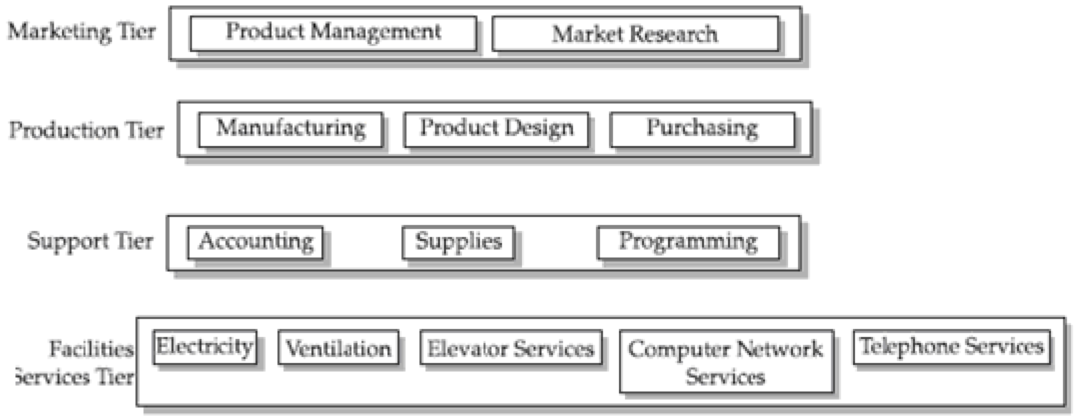


Figure 1.3 Resources of a large organization are typically organized into a tier structure that operates similarly to the tier structure used in distributed systems.

At the lowest level of a corporation are facilities services that consist of resources necessary to maintain the office building. Facilities services encompass a wide variety of resources that typically include electricity, ventilation, elevator services, computer network services, and telephone services. The next tier in the organization contains support resources such as accounting, supplies, computer programming, and other resources that support the main activity of the company. Above the support tier is the production tier. The production tier has the resources necessary to produce products and services sold by the company. The highest tier is the marketing tier, which consists of resources used to determine the products and services to sell to customers.

Any resource is considered a client when a resource sends a request for service to a service provider (also referred to as a service). A serv.ice is any resource that receives and fulfills a request from a client, and that resource itself might have to make requests to other resources to fulfill a client's request. For Example a product manager working at the marketing tier decides the company could make a profit by selling customers a widget. The product manager requests an accountant to conduct a formal cost analysis of manufacturing a widget.

The accountant is on the support tier of the organization. The product manager is the client and the accountant is the service. However, the accountant requires information from the manufacturing manager to fulfill the product manager's request. The manufacturing manager works on the production tier of the organization. The accountant is the client to the manufacturing manager who is the service to the accountant. In multi-tier architecture, each tier contains services that include software objects, database management systems (DBMS), or connectivity to legacy systems.

Information technology departments of corporations employ multi-tier architecture because it's a cost-efficient way to build an application that is flexible, scalable, and responsive to the expectations of clients. This is because the functionality of the application is divided into logical components that are associated with a tier. Each component is a service that is built and maintained independently of other services. Services are bound together by a communication protocol that enables a service to receive and send information from and to other services.

A client is concerned about sending a request for service and receiving results from a service. A client isn't concerned about how a service provides the results. This means that a programmer can quickly develop a system by creating a client program that formulates requests for services that already exist in the multi-tier architecture. These services already have the functionality built into them to fulfill the request made by the client program.

Services can be modified as changes occur in the functionality without affecting the client program. For example, a client might request the tax owed on a specific order. The request is sent to a service that has the functionality to determine the tax. The business logic for calculating the tax resides within the service. A programmer can modify the business logic in the service to reflect the latest changes in the tax code without having to modify the client program. These changes are hidden from the client program.

J2EE Multi-Tier Architecture

J2EE is four-tier architecture (see Figure1.4). These consist of the Client Tier (sometimes referred to as the Presentation Tier or Application Tier), Web Tier, Enterprise JavaBeans Tier (sometimes referred to as the Business Tier), and the Enterprise Information Systems Tier. Each tier is focused on providing a specific type of functionality to an application.It's important to delineate between physical location and functionality. 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. And since the J2EE multi-tier architecture is functionally centric, a J2EE application accesses only tiers whose functionality is required by the J2EE application. It's also important to disassociate a J2EE API with a particular tier. That is, some APIs (i.e., XML API) and

J2EE components can be used on more than one tier, while other APIs (i.e., Enterprise JavaBeans API) are associated with a particular tier. The Client Tier consists of programs that interact with the user. These programs prompt the user for input and then convert the user's response into requests that are forwarded to software on a component that processes the request and returns results to the client program. The component can operate on any tier, although most requests from clients are processed by components on the Web Tier. The client program also translates the server's response into text and screens that are presented to the user.

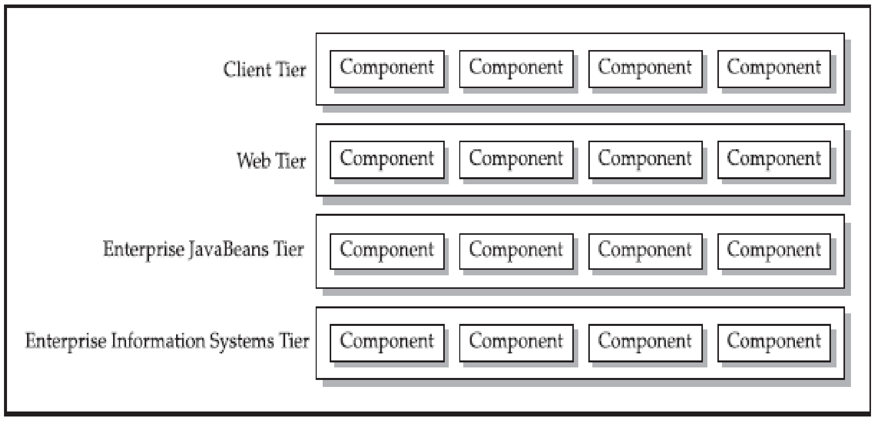


Figure1.4 J2EE consists of four tiers, each of which focuses on providing specific functionality to an application.

The Web Tier provides Internet functionality to a J2EE application. Components that operate on the Web Tier use HTTP to receive requests from and send responses to clients that could reside on any tier. A client is any component that initiates a request, as explained previously in this chapter. For example (see Figure 1.5), a client's request for data that is received by a component working on the Web Tier is passed by the component to the Enterprise JavaBeans Tier where an Enterprise Java Bean working on the Enterprise JavaBeans

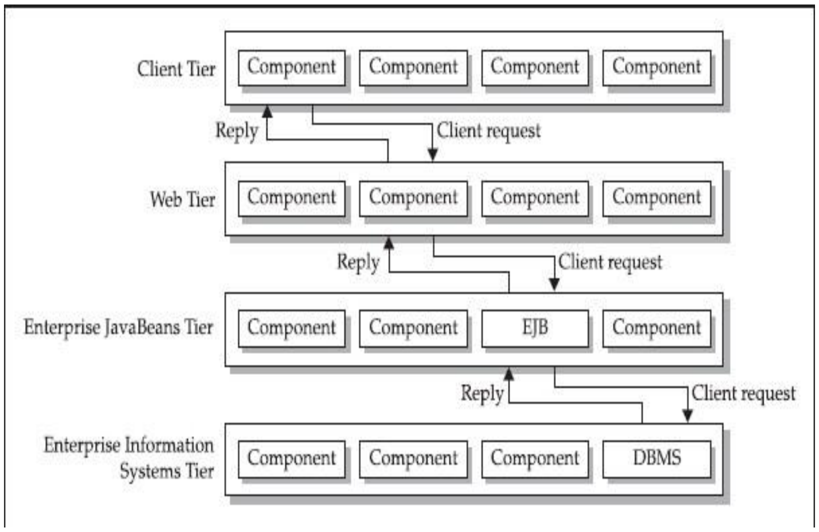


Figure 1.5 J2EE consists of four tiers, each of which focuses on providing specific functionality to an application

A request is typically passed from one tier to another before the Tier interacts with DBMS to fulfill the request. Requests are made to the Enterprise JavaBeans by using the Java Remote Method Invocation (RMI) API. The requested data is then returned by the Enterprise JavaBeans where the data is then forwarded to the Web Tier and then relayed to the Client Tier where the data is presented to the user. The Enterprise JavaBeans Tier contains the business logic for J2EE applications.
. 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 and because ACL 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. The EIS links a J2EE application to resources and legacy systems that are available on the corporate backbone network. It's on the EIS where a J2EE application directly or indirectly interfaces with a variety of technologies, including DBMS and mainframes that are part of the mission-critical systems that keep the corporation operational. Components that work on the EIS communicate to resources using CORBA or Java connectors, referred to as J2EE Connector Extensions.

CLIENT TIER IMPLEMENTATION

There are two components on the Client Tier that are described in the J2EE specification. These are applet clients and application clients. An applet client is a component used by a web client that operates within the applet container, which is a Java-enabled browser. An applet uses the browser as a user interface.

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. A rich client is a third type of client, but a rich client is not considered a component of the Client Tier because a rich client can be written in a language other than Java and therefore J2EE doesn't define a rich client container.
A rich client is similar to an application client in that both are applications that contain their own user interface. And as with an application client, a rich client can access any tier in the environment, depending on the ACLs configuration, using HTTP, SOAP, ebXML, or an appropriate protocol.

WEB TIER IMPLEMENTATION

The Web Tier has several responsibilities in the J2EE multi-tier architecture, all of which is provided to the Client Tier using HTTP. These responsibilities are to act as an intermediary between components working on the Web Tier and other tiers and the Client Tier.

Intermediary activities include:

- Accepting requests from other software that was sent using POST, GET, and PUT operations, which are part of HTTP transmissions

- Transmit data such as images and dynamic content

There are two types of components that work on the Web Tier. These are servlets and Java Server Pages (JSP), although many times they are proxied to the Application or EJB Tier. A servlet is a Java class that resides on the Web Tier and is called by a request from a browser client that operates on the Client Tier. A servlet is associated with a URL that is mapped by the servlet container.

A request for a servlet contains the servlet's URL and is transmitted from the Client Tier to the Web Tier using HTTP. The request generates an instance of the servlet or reuses an existing instance, which receives any input parameters from the Web Tier that are necessary for the servlet to perform the service. Input parameters are sent as part of the request from the client.

An instance of a servlet fulfills the request by accessing components/resources on the Web Tier or on other tiers as is necessary based on the business logic that is encoded into the servlet. The servlet typically generates an HTML output stream that is returned to the web server. The web server then transmits the data to the client. This output stream is a dynamic web page.

JSP is similar to a servlet in that a JSP is associated with a URL and is callable from a client. However, JSP is different than a servlet in several ways, depending on the container that is used. Some containers translate the JSP into a servlet the first time the client calls the JSP, which is then compiled and the compiled servlet loaded into memory. The servlet remains in memory. Subsequent calls by the client to the JSP cause the web server to recall the servlet without translating the JSP and compiling the resulting code. Other containers precompile a JSP into a .java file that looks like a servlet file, which is then compiled into a Java class.

Business logic used by JSP and servlet's is contained in one or more Enterprise JavaBeans that are callable from within the JSP and servlet. The code is the same for both JSP and servlet, although the format of the code differs. JSP uses custom tags to access an Enterprise JavaBeans while servlet's are able to directly access Enterprise JavaBeans.

ENTERPRISE JAVABEANS TIER IMPLEMENTATION

J2EE uses distributive object technology to enable Java developers to build portable, scalable, and efficient applications that meet the 24-7 durability expected from an enterprise system. The Enterprise JavaBeans Tier contains the Enterprise JavaBeans server, which is the object server that stores and manages Enterprise JavaBeans. The Enterprise JavaBeans Tier is a vital element in the J2EE multi-tier architecture because this tier provides concurrency, scalability, lifecycle management, and fault tolerance. The Enterprise JavaBeans Tier automatically handles concurrency issues that assure multiple clients have simultaneous access to the same object. The Enterprise JavaBeans Tier is the tier where some vendors include features that enable scalability of an application, because the tier is designed to work in a clustered environment. This assumes that vendor components that are used support clustering. If not, a Local Director is typically used for horizontal load balancing

The Enterprise JavaBeans Tier manages instances of components. This means component containers working on the Enterprise JavaBeans Tier create and destroy instances of components and also move components in and out of memory. Fault-tolerance is an important consideration in mission-critical applications. The Enterprise JavaBeans Tier is the tier where some vendors include features that provide fault-tolerant operation by making it possible to have multiple Enterprise JavaBeans servers available through the tier. This means backup Enterprise JavaBeans servers can be contacted immediately upon the failure of the primary Enterprise JavaBeans server. The Enterprise JavaBeans server has an Enterprise JavaBeans container within which is a collection of Enterprise JavaBeans. As discussed in previous sections of this chapter, an Enterprise Java Bean is a class that contains business logic and is callable from a servlet or JSP.

Collectively 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.

These system services are

- Resource pooling
- Distributed object protocols
- Thread management
- State management
- Process management
- Object persistence
- Security
- Deploy-time configuration

A key benefit of using the Enterprise JavaBeans server and Enterprise JavaBeans container technology is that this technology makes proper use of a programmer's expertise. That is, a programmer who specializes in coding business logic isn't concerned about coding system services. Likewise, a programmer whose specialty is system services can focus on developing system services and not be concerned with coding business logic.

Any component, regardless of the tier where the component is located, can use Enterprise JavaBeans. This means that an Enterprise Java Bean client can reside outside the Client Tier. The protocol used to communicate between the Enterprise JavaBeans Tier and other tiers is dependent on the protocol used by the other tier. 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. Sometimes software on other tiers, usually the middle tier, uses JMS to communicate with the Enterprise JavaBeans Tier.

This communication isn't exclusively used to send and receive messages between machines. JMS is also used for other communication, such as decoupling tiers using the queue mechanism. However, the Enterprise Java Bean that is used must be a message-driven bean (MDB). MDBs are commonly used to process messages on a queue that may or may not reside on the local machine.

Enterprise Information Systems Tier Implementation

The Enterprise Information Systems (EIS) Tier is the J2EE architecture's connectivity to resources that are not part of J2EE. These include a variety of resources such as legacy systems, DBMS, and systems provided by third parties that are accessible to components in the J2EE infrastructure. 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. Likewise, developers can utilize off-the-shelf software that is commercially available in the marketplace because the 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. Java Connector technology enables software developers to create a Java Connector for legacy systems and for third-party software. The connector defines all the elements that are needed to communicate between the J2EE application and the non-J2EE software. This includes rules for connecting to each other and rules for conducting secured transactions.

J2EE COMPONENTS

J2EE applications are made up of components. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components.
The J2EE specification defines the following J2EE components:

- Application clients and applets are components that run on the client.
- Java Servlet and Java Server Pages (JSP) technology components are Web components that run on the server.

Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server.

J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and ―standard‖ Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

J2EE Clients

A J2EE client can be a Web client or an application client.

Web Clients

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 renders the pages received from the server.

A Web client is sometimes called a thin client. Thin clients usually do not do things like query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, heavyweight operations like these are off-loaded to enterprise beans executing on the J2EE server where they can leverage the security, speed, services, and reliability of J2EE server-side technologies.

Applets

A Web page received from the Web tier can include an embedded applet. 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. However, client systems will likely need the Java Plug-in and possibly a security policy file in order for the applet to successfully execute in the Web browser. Web components are the preferred API for creating a Web client program because no plug-ins or security policy files are needed on the client systems. Also, Web components enable cleaner and more modular application design because they provide a way to separate applications programming from Web page design. Personnel involved in Web page design thus do not need to understand Java programming language syntax to do their jobs.

Application Clients

A J2EE 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 Swing or Abstract Window Toolkit (AWT) APIs, but a command-line interface is certainly possible. Application clients directly access enterprise beans running in the business tier. However, if application requirements warrant it, a J2EE application client can open an HTTP connection to establish communication with a servlet running in the Web tier.

KEY TERMS

- **Java bytecode:** It is the form of instructions that the Java virtual machine executes
- **Java 2 Platform, Standard Edition (J2SE):** It is used primarily for writing applets and other Java-based applications. One of the primary uses of J2SE is the development of Java applications for individual computers.
- **Multi-Tier Architecture**: It is a client-server architecture in which the presentation, the application processing, and the data management are logically separate processes
- **Client Tier:** In the client tier, Web components, such as Servlets and Java Server Pages (JSPs), or standalone Java applications provide a dynamic interface to the middle tier.
- **Middle Tier**: In the server tier, or middle tier, enterprise beans and Web Services encapsulate reusable, distributable business logic for the application. These server-tier components are contained on a J2EE Application Server, which provides the platform for these components to perform actions and store data.
- **Enterprise Data Tier**: In the data tier, the enterprise's data is stored and persisted, typically in a relational database.
- **Application client:** A first-tier J2EE client component that executes in its own Java virtual machine. Application clients have access to some J2EE platform APIs.

QUESTIONS

Short Answer Questions

1. Differentiate JS2E and J2EE
2. Define Java ByteCode. List the advantages of Java
3. What is Distributive Systems?
4. What is J2EE Components? List the Components
5. What is a Tier architecture?
6. Write short notes on Clients , Resources and Components in Multi-tier arch

Long –Answer Question

1. What is Java? Describe the advantages of Java
2. Describe in Detail Various J2EE Tier Architecture
3. Explain Enterprise Java Beans and Enterprise Information Systems Tier Implementation in detail

References

1. Jim Keogh , ―The Complete Reference J2SE‖, 1st Edition, Tata McGraw Hill Edition, New Delhi:2002.
2. Joseph j Bambara et al, :J2EE Unleashed‖ , 1st Edition, Tech Media, 2001
3. J2EE Tutorial - http://java.sun.com/j2ee/tutorial/1_3-fcs
4. J2EE Developers Guide - http://java.sun.com/j2ee/sdk_1.2.1/techdocs/guides/ejb/html/DevGuideTOC.html

UNIT-1
J2EE

S. No.	**Questions**	**Option 1**	**Option 2**	**Option 3**	**Option 4**	**Answer**
1	The expansion of BCPL is __________.	Basic Combined Programming Language	Beginners Combined Programming Language	Basic Control Programming Language	Beginners Control Programming Language	Basic Combined Programming Language
2	Programmers divide a program into functionality and create code segments called ___________.	programs	subprograms	macros	functions	functions
3	In the year __________ the American National Standard Institute formally adopted a standard for the C Programming language.	1970	1980	1990	2000	1990
4	Java is ___________ programming language that uses classes to create instances of objects.	object based	object oriented	procedure oriented	knowledge based	Object oriented
5	__________ is a routine that recovers spent memory without the programmer having to write code to free previously reserved memory.	Memory release	Garbage collection	Memory management	Garbage compaction	Garbage collection
6	Java __________converts java source code into byte code that is executed by the Java Virtual machine.	interpreter	compiler	assembler	preprocessor	compiler
7	Java compiler generates ___________.	binary code	octal code	byte code	hexadecimal code	byte code
8	Small amount of data stored on the client is called ___________.	cookie	servlet	images	applet	cookie
9	An __________ is a small program that can be efficiently downloaded over the internet and is executed by a java compatible browser.	cookie	servlet	images	applet	applet

UNIT-1
J2EE

10	Request and execution occur on the user’s computer called __________.	server	client	client and server	JVM	client
11	Embedded in the web page might be a reference to run a small java program called an ___________.	cookie	applet	image	servlet	applet
12	The __________ reads the reference to the applet, then requests that the web server download the applet.	cookie	browser	servlet	applet	browser
13	Once the applet is received, the browser requests the _______ to execute the applet automatically without any additional interaction by the user.	server	client	client and server	JVM	JVM
14	__________ could not offer the dynamics demanded by internet users and corporations.	Static web pages	Dynamic web pages	Browsers	Applets	Static web pages
15	Java was developed by ___________.	IBM	Microsoft	Sun Microsystems	Oracle Corporation	Sun Microsystems
16	Features found in ___________ were adopted in Java by the Java development team.	C only	C++ only	C and C++	Visual C++	C and C++
17	Java is a __________ programming language.	multiuser	multitasking	multithreaded	procedure oriented	multithreaded
18	A _________ is a process that can work independently from other processes and permit multiple access to the same program simultaneously.	macro	procedure	function	thread	thread
19	The original edition of Java is called ___________.	J2ME	J2SE	J2EE	Core Java	J2SE
20	A __________ program is automatically translated into a java servlet.	Java	EJB	JSP	d) HTML	JSP
21	__________ interfaces between commercial DBMS products and Java.	API	EJB	JSP	XML	API

22	__________ contains the API used to create wireless java applications.	J2ME	J2SE	J2EE	EJB	J2ME
23	During the evolutionary process, the java development team included more interfaces and libraries as programmers demanded new APIs. These new features to the JDK were called __________.	SDK	Java Bean	BDK	Extensions	Extensions
24	__________ consists of specifications and API for developing reusable server-side business components designed to run on applications servers.	Java	EJB	JSP	Servlets	EJB
25	__________ is a program that resides on the server	. Servlet	Cookie	Applet	JSP	Servlet
26	__________ consists of specifications and APIs for developing reusable server-side business components designed to run on applications servers.	EJB	JSP	Servlets	Java	EJB
27	A __________ bean retains data accumulated during a session with a client.	session servlet	stateful session	stateless session	JMS container	stateful session
28	A __________ bean does not maintain any state between method calls.	session servlet	stateless session	stateful session	JMS container	stateless session
29	A message-driven bean is called by the __________.	sessionservlet	JMS container	message-oriented middleware	API	JMS container
30	The core components of J2EE are __________.	Java Beans	Java servlets and Java beans	Java servlets and JSPs	Java beans, Java servlets and JSPs	Java beans, Java servlets and JSPs
31	The expansion of CORBA is __________.	Combined Object Request Basic Architecture	Common Object Request Broker Architecture	Combined Object Request Broker Architecture	Common Object Request Basic Architecture	Common Object Request Broker Architecture
32	The expansion of XDR is __________.	Exchange Data Representation	External Data Representation	External Digital Representation	Experimental Data	External Data Representation

					Representation	
33	_________ are the internal software services.	servlets	functions	RPCs	JSPs	functions
34	_________ are the external software services.	servlets l	functions	RPCs	JSPs	RPCs
35	In multi-tier architecture, each tier contains _________ that include software objects, DBMS or connectivity to legacy systems.	services	java programs	servlets	requests	services
36	___________ is a part of a tier that consists of a collection of classes or a program that performs a function to provide the services.	container	component	resource	service	component
37	A _________ is anything a component needs to provide a service.	container	component	resource	service	resource
38	A __________ is a software that manages a component and provides a component with system services.	container	component	resource	service	container
39	J2EE is a _________ tier architecture.	2	3	4	5	4
40	_________ tiers can physically reside on the same JVM although each tier provides a different type of functionality to a J2EE application.	1	2	3	4	3
41	The _________ tier consists of programs that interact with the user.	client	web	EJB tier	EIS	client
42	The ________ provides internet functionality to a J2EE application.	client	web	EJB tier	EIS	web
43	The ________ tier contains the business logic for J2EE applications.	client	web	EJB tier	EIS	EJB tier

UNIT-1
J2EE

44	The _______ tier links a J2EE application to resources and legacy systems that are available on the corporate backbone network.	client	web	EJB tier	EIS	EIS
45	The _________ tier is the keystone to every J2EE application.	client	web	EJB tier	EIS	EJB tier
46	__________ are contained on the EJB server which is a distributed object server that works on the EJB tier.	servlets	EJB	JSP	client programs	EJB
47	It is on the __________ where a J2EE application directly or indirectly interfaces with a variety of technologies including DBMS and mainframes.	servlets	EJB	JSP	client programs	client programs
48	There are ___________ components on the client tier.	2	3	4	5	2
49	A ______ is a component used by a web client that operates within the applet container, which is a java-enabled browser.	application client	applet client	servlet	JSP	applet client
50	A ________ is a java application that operates within the application client container, which is the Java 2 Runtime Environment Standard Edition.	application client	applet client	servlet	JSP	application client
51	A ________ has its own interface and is capable of accessing all the tiers in the multi-tier architecture.	application client	applet client	application	servlet	application
52	A __________ is not considered as the component of the client tier.	application client	applet client	rich client	server	rich client
53	A __________ can access any tier in the environment depending on the ACLs configuration using HTTP, SOAP, etc.	application client	applet client	rich client	servlet	rich client
54	Clients are classified into ___________	2	3	4	5	4

	types.					
55	A __________ consists of software usually a browser that accesses resources located on the web tier.	web client	EJB client	EIS client	multitier client	web client
56	____________ only accesses one or more EJB that are located on the EJBs tier rather than resources on the web tier.	web client	EJB client	EIS client	multitier client	EJB client
57	__________ are the interface between users and resources located on the EIS tier.	web client	EJB client	EIS client	multitier client	EIS client
58	A __________ is a unique type of client because it is also a service that works on the web tier.	web client	EJB client	EIS client	web service peer	web service peer
59	___________ are conceptually similar to a web service peer.	web client	EJB client	EIS client	multitier client	multitier client
60	___________ are similar to web clients.	web client	EJB client	EIS client	multitier client	2

UNIT II
J2EE DATABASE CONCEPTS

ABOUT DATA

The term data means groups of information that represent the qualitative or quantitative attributes of a variable or set of variables. Data (plural of "datum") are typically the results of measurements and can be the basis of graphs, images, or observations of a set of variables. Data are often viewed as the lowest level of abstraction from which information and knowledge are derived. In computer science, data is anything in a form suitable for use with a computer. Data is often distinguished from programs. Data is a collection of facts, figures and statistics related to an object. Data can be processed to create useful information. Data is a valuable asset for an organization. Data can be used by the managers to perform effective and successful operations of management. It provides a view of past activities related to the rise and fall of an organization. It also enables the user to make better decision for future. Data is very useful for generating reports, graphs and statistics.

Example:
Students fill an admission form when they get admission in college. The form consists of raw facts about the students. These raw facts are student's name, father name, address etc. The purpose of collecting this data is to maintain the records of the students during their study period in the college.

ABOUT DATABASE

A database is an integrated collection of logically related records or files consolidated into a common pool that provides data for one or more multiple uses. One way of classifying databases involves the type of content, for example: bibliographic, full-text, numeric, and image. Software organizes the data in a database according to a database model. A number of database architectures exist. Many databases use a combination of strategies. Databases consist of software-based "containers" that are structured to collect and store information so users can retrieve, add, update or remove such information in an automatic fashion. Database programs are designed for users so that they can add or delete any information needed. The structure of a database is the table, which consists of rows and columns of information.

DATABASE SCHEMA

The schema of a database system is its structure described in a formal language supported by the database management system (DBMS). In a relational database, the schema defines the tables, the fields, relationships, views, indexes, packages, procedures, functions, queues, triggers, types, sequences, materialized views, synonyms, database links, directories, Java, XML schemas, and other elements. Schemas are generally stored in a data dictionary. Although a schema is defined in text database language, the term is often used to refer to a graphical depiction of the database structure.
Levels of database schema

- Conceptual schema, a map of concepts and their relationships.
- Logical schema, a map of entities and their attributes and relations
- Physical schema, a particular implementation of a logical schema

- Schema object, Oracle database object

Conceptual schema

A conceptual schema or conceptual data model is a map of concepts and their relationships. This describes the semantics of an organization and represents a series of assertions about its nature. Specifically, it describes the things of significance to an organization (entity classes), about which it is inclined to collect information, and characteristics of (attributes) and associations between pairs of those things of significance (relationships).

Logical schema

A Logical Schema is a data model of a specific problem domain expressed in terms of a particular data management technology. Without being specific to a particular database management product, it is in terms of relational tables and columns, object-oriented classes, or XML tags. This is as opposed to a conceptual data model, which describes the semantics of an organization without reference to technology, or a physical data model, which describe the particular physical mechanisms used to capture data in a storage medium. The next step in creating a database, after the logical schema is produced, is to create the physical schema.

Physical Schema

Physical Schema is a term used in relation to data management. In the ANSI four-schema architecture, the internal schema was the view of data that involved data management technology. This was as opposed to the external schema that reflected the view of each person in the organization, or the conceptual schema that was the integration of a set of external schemas.

Schema Object

A schema object is a logical data storage structure. Schema objects do not have a one-to-one correspondence to physical files on disk that store their information. However, Oracle stores a schema object logically within a table space of the database. The data of each object is physically contained in one or more of the table space's data files. For some objects such as tables, indexes, and clusters, you can specify how much disk space Oracle allocates for the object within the table space's data files.

There is no relationship between schemas and table spaces: a table space can contain objects from different schemas, and the objects for a schema can be contained in different table spaces.

Associated with each database user is a schema. A schema is a collection of schema objects. Examples of schema objects include tables, views, sequences, synonyms, indexes, clusters, database links, snapshots, procedures, functions, and packages.

DATABASE AND PLATFORM PORTABILITY

Data connectivity architecture can either simplify or radically complicate portability among databases, database versions, and platforms. Ideally, data connectivity components should share a common architecture that makes it easy to change or upgrade the underlying database infrastructure. Most software companies and enterprise IT organizations must support more than one database platform – and more than one version of every platform they support. This can mean also managing a myriad of data connectivity methods, driver versions, and client library packages.

Adding a new database or even upgrading to a new version of the same database can create substantial development, integration, and testing work. For example, data connectivity components designed to work with only one database will handle BLOB/CLOB data types (large binary or character objects) differently than a component designed to work exclusively with another database. Developers will spend significant time and effort on additional coding and testing for each new database that they need to support. Standardizing and simplifying data connectivity architecture dramatically reduces the cost and complexity associated with supporting multiple database back ends. For independent software vendors in particular, this is a significant business priority.

INTRODUCTION TO JDBC

An application programming Interface (API) is a set of classes, methods and resources that programs can use to do their work. APIs exist for windowing systems, file systems, database systems, networking systems, and others. JDBC is a Java API for database connectivity that is part of the Java API developed by Sun Microsystems. JDBC provides Java developers with an industry standard API for database-independent connectivity between java applications and a wide range of relational database management systems such as oracle. Informix, Microsoft SQL Server and Sybase.

The API provides a call level interface to the database.

- Connect to a database
- Execute SQL statements to query your database
- Generate query results
- Perform updates, inserts and deletions
- Execute stored procedures

The following figure 2.5.1 shows the components of the JDBC model. In its simplest form, JDBC makes it possible to do these basic things: The Java application calls JDBC classes and interfaces to submit SQL statements and retrieve results.

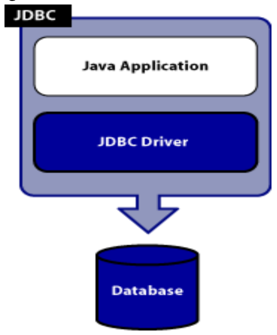


Figure 2.5.1 Components of the JDBC Model

The JDBC API is implemented through the JDBC driver. The JDBC Driver is a set of classes that implement the JDBC interfaces to process JDBC calls and return result sets to a Java application. The database (or data store) stores the data retrieved by the application using the JDBC Driver.

JDBC OBJECTS

The main objects of the JDBC API include:

- A Data Source object is used to establish connections. Although the Driver Manager can also be used to establish a connection, connecting through a Data Source object is the preferred method.
- A Connection object controls the connection to the database. An application can alter the behavior of a connection by invoking the methods associated with this object. An application uses the connection object to create statements.
- Statement, Prepared Statement, and Callable Statement objects are used for executing SQL statements. A Prepared Statement object is used when an application plans to reuse a statement multiple times. The application prepares the SQL it plans to use. Once prepared, the application can specify values for parameters in the prepared SQL statement. The statement can be executed multiple times with different parameter values specified for each execution. A Callable Statement is used to call stored procedures that return values. The Callable Statement has methods for retrieving the return values of the stored procedure

A ResultSet object contains the results of a query. A ResultSet is returned to an application when a SQL query is executed by a statement object. The ResultSet object provides methods for iterating through the results of the query

BENEFITS OF JDBC

The benefits of using JDBC include the following:

- A developer only needs to know one API to access any relational database
- There is no need to rewrite code for different databases.
- There is no need to know the database vendor's specific APIs
- It provides a standard API and is vendor independent
- Almost every database vendor has some sort of JDBC driver
- JDBC is part of the standard Java 2 platform

JDBC ARCHITECTURE

The JDBC API contains two major sets of interfaces: the first is the JDBC API for application writers, and the second is the lower-level JDBC driver API for driver writers. JDBC technology drivers fit into one of four categories. Applications and applets can access databases via the JDBC API using pure Java JDBC technology-based drivers, as shown in the Figure 2.7.1 below

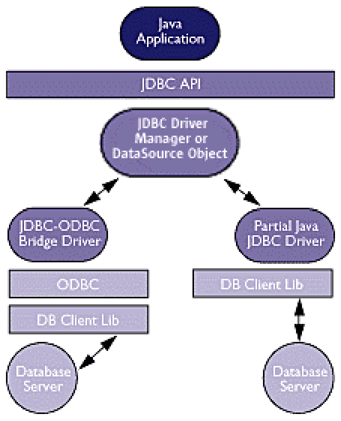
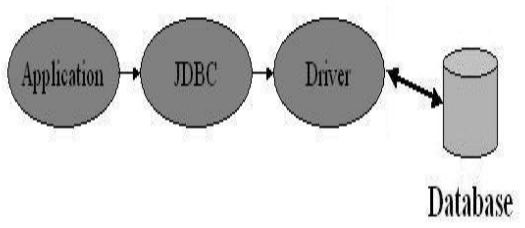


Figure 2.7.1 JDBC connectivity using ODBC drivers

Left side, Type 1: JDBC-ODBC Bridge plus ODBC Driver
This combination provides JDBC access via ODBC drivers. ODBC binary code and in many cases, database client code must be loaded on each client machine that uses a JDBC-ODBC Bridge. Sun provides a JDBC-ODBC Bridge driver, which is appropriate for experimental use and for situations in which no other driver is available.
Right side, Type 2: A native API partly Java technology-enabled driver. This type of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.
Right side, Type 3: Pure Java Driver for Database Middleware
This style of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.
Left side, Type 4: Direct-to-Database Pure Java Driver
This style of driver converts JDBC calls into the network protocol used directly by DBMS, allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access.

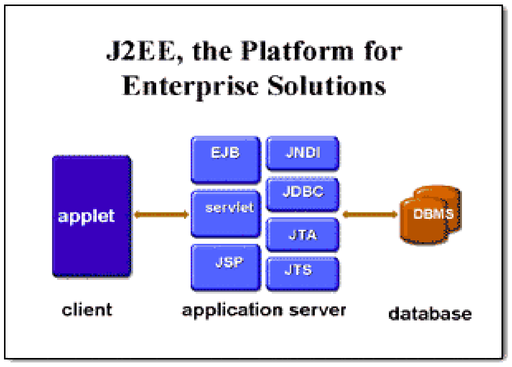
Java application calls the JDBC library. JDBC loads a driver which talks to the database. We can change database engines without changing database code. The Figure 2.7.2 shows the architecture of JDBC.



.

Figure 2.7.2 JDBC Architecture

JDBC IN J2EE
As a core part of the Java 2 Platform, the JDBC API is available anywhere that the platform is. This means that your applications can truly write database applications once and access data anywhere. The JDBC API is included in the Java 2 Platform, Standard Edition (J2SE) and the Java 2 Platform, Enterprise Edition (J2EE), providing server-side functionality for industrial strength scalability.
An example of a J2EE based architecture that includes a JDBC implementation:



Requirements

Software: The Java 2 Platform (either the Java 2 SDK, Standard Edition, or the Java 2 SDK, Enterprise Edition), an SQL database, and a JDBC technology-based driver for that database.
Hardware: Same as for the Java 2 Platform.

TWO-TIER AND THREE-TIER MODELS

The JDBC API supports both two-tier and three-tier models for database access Fig 2.8.1 illustrate two-tier architecture for data access

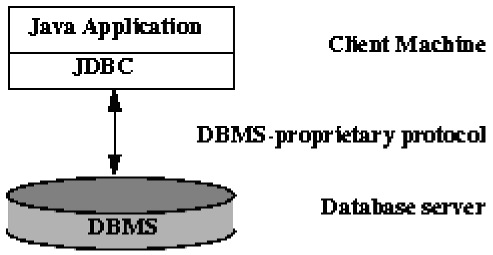


Fig 2.8.1 Two tier architecture for data access

In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet**.**
In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages. In Figure 2.8.2: illustrates three-tier architecture for database access.

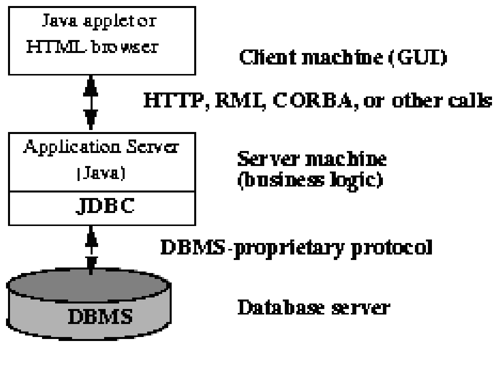


Fig 2.8.2 Three-tier architecture for database access

Until recently, the middle tier has typically been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java bytecode into efficient machine-specific code and technologies such as Enterprise JavaBeans, the Java platform is fast becoming the standard platform for middle-tier development. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features.
With enterprises increasingly using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed transactions, and disconnected rowsets.

DRIVER TYPES

JDBC technology-based drivers generally fit into one of four categories. In Figure shows various driver implementation possibilities

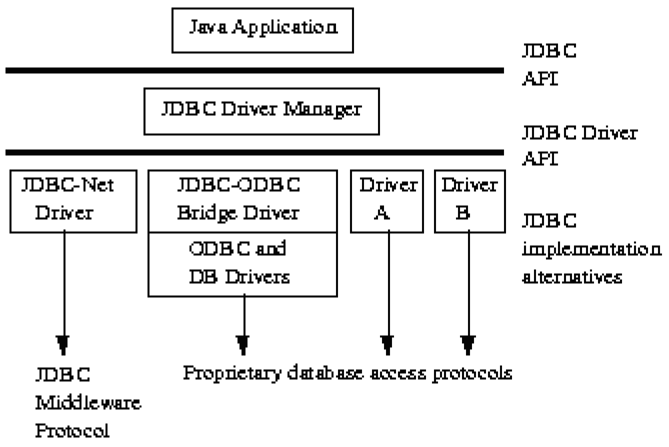


Figure 2.9.1 Various driver implementation possibilities

JDBC technology-based drivers generally fit into one of four categories. In Figure shows various driver implementation possibilities

JDBC Drivers Types

Sun has defined four JDBC driver types. These are:

Type 1: JDBC-ODBC Bridge Driver

The first type of JDBC driver is JDBC-ODBC Bridge which provides JDBC access to any ODBC complaint databases through ODBC drivers. Sun's JDBC-ODBC bridge is example of type 1 driver.

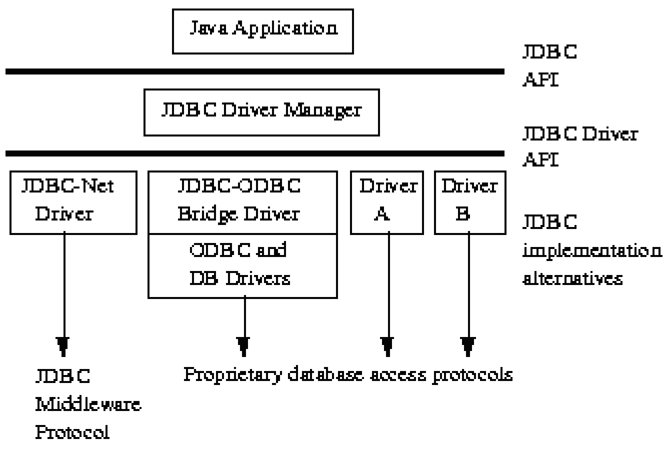


Figure 2.9.2 Various driver implementation possibilities

Type 2: Native -API Partly - Java Driver

Type 2 drivers are developed using native code libraries, which were originally designed for accessing the database through C/C++. Here a thin code of Java wrap around the native code and converts JDBC commands to DBMS-specific native calls.

Type 3: JDBC-Net Pure Java Driver

Type 3 drivers are three-tier solutions. This type of driver communicates to a middleware component which in turn connects to database and provide database connectivity.

Type 4: Native-Protocol Pure Java Driver

Type 4 drivers are entirely written in Java that communicates directly with vendor's database through socket connection. Here no translation or middleware layer, are required which improves performance tremendously

JDBC-ODBC Bridge driver *(Type 1 JDBC Driver)*

The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available. In figure 2.10.1 Type 1 JDBC – ODBC Bridge.

Advantage

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

Disadvantages

1. Since the Bridge driver is not written fully in Java, Type1 drivers are not portable
2. A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.

The client system requires the ODBC Installation to use the driver and Not good for the Web.

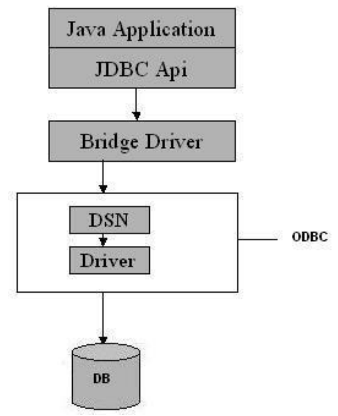


Figure 2.10.1 Type **1: JDBC-ODBC Bridge**

Native-API/partly Java driver (Type 2 JDBC Driver)

The distinctive characteristic of type 2 jdbc drivers is that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.

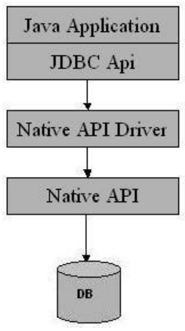


Figure 2.10.2 Type 2: Native API/ Partly Java Driver

<u>Advantage</u>

The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type 1 and also it uses Native api which is Database specific.

<u>Disadvantage</u>

1. Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
2. Like Type 1 drivers, it’s not written in Java Language which forms a portability issue.
3. If we change the Database we have to change the native api as it is specific to a database

4. Mostly obsolete now
5. Usually not thread safe.

All Java/Net-protocol driver (Type 3 JDBC Driver)

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers.

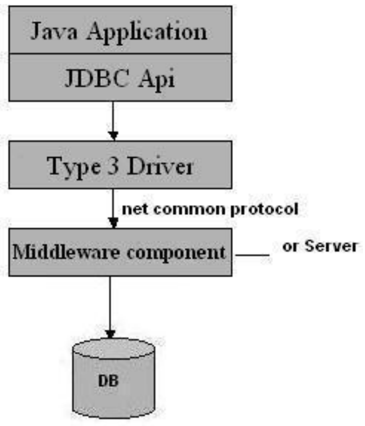


Figure 2.10.3 **Type 3: All Java/ Net-Protocol Driver**

<u>**Advantage**</u>

1. This driver is server-based, so there is no need for any vendor database library to be present on client machines.
2. This driver is fully written in Java and hence Portable. It is suitable for the web

There are many opportunities to optimize portability, performance, and scalability.

3. The net protocol can be designed to make the client JDBC driver very small and fast to load.
4. The type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.
5. This driver is very flexible allows access to multiple databases using one driver
6. They are the most efficient amongst all driver types.

<u>**Disadvantage**</u>

It requires another server application to install and maintain. Traversing the recordset may take longer, since the data comes through the backend server

Native-protocol/all-Java driver (Type 4 JDBC Driver)

The Type 4 uses java networking libraries to communicate directly with the database server.

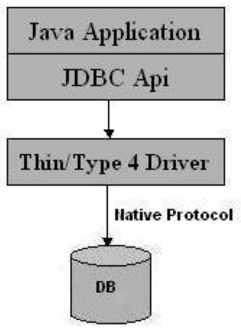


Figure 2.10.4 Type 4: Native-protocol/all-Java driver

Advantage

1. The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
2. Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good
3. You don’t need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantage

With type 4 drivers, the user needs a different driver for each database.

JDBC PACKAGE

The purpose of the JDBC package is to provide vendor-neutral access to relational databases. The implementation differences of the various databases used are abstracted from the user through the use of the JDBC API. Though the specification does not indicate that the API is to be used solely for relational databases, historically it has been used primarily for relational database access.

The developers of the JDBC API specification have tried to keep the API as simple as possible so that it can be a foundation upon which other APIs are built. For instance, the connector API can be implemented on top of an existing JDBC API using appropriate resource adapters. JDBC is composed of a number of interfaces. These interfaces are implemented by driver developers. The API is implemented by either a vendor or a third party to create a JDBC driver.

The Type 4 JDBC driver is considered the best driver to use for two reasons. One reason is that since the driver has been written completely in Java, it is extremely portable. Another reason is that the driver is not required to map JDBC calls to corresponding native CLI calls. This avoids the overhead of mapping logic required by the Type 1 or Type 2 driver, or the overhead of communicating with middleware required by the Type 3 driver.

Such improvements in efficiency should allow the driver to execute faster than the other types of JDBC drivers.

JDBC 2.0 API

The JDBC 2.0 API includes the complete JDBC API, which includes both core and Optional Package API, and provides industrial-strength database computing capabilities. It is not, however, limited to SQL databases; the JDBC 2.0 API makes it possible to access data from virtually any data source with a tabular format.

The JDBC 2.0 API includes two packages:

- java.sql package--the JDBC 2.0 core API
  - JDBC API included in the JDKTM 1.1 release (previously called JDBC 1.2). This API is compatible with any driver that uses JDBC technology.
  - JDBC API included in the Java 2 SDK, Standard Edition, version 1.2 (called the JDBC 2.0 core API). This API includes the JDBC 1.2 API and adds many new features.
- javax.sql package--the JDBC 2.0 Optional Package API. This package extends the functionality of the JDBC API from a client-side API to a server-side API, and it is an essential part of Java2 SDK, Enterprise Edition technology.

  Being an Optional Package, it is not included in the Java 2 Platform SDK, Standard Edition, version 1.2, but it is readily available from various sources.

  - Information about the JDBC 2.0 Optional Package API is available from the JDBC web page. The `javax.sql` package may also be downloaded from this web site.
  - Driver vendors may include the `javax.sql` package with their products.
  - The Java 2 SDK, Enterprise Edition, includes many Optional Package APIs, including the JDBC 2.0 Optional Package.

The java.sql Package

The `java.sql` package contains the entire JDBC API that sends SQL (Structured Query Language) statements to relational databases and retrieves the results of executing those SQL statements.

The `Driver` interface represents a specific JDBC implementation for a particular database system. `Connection` represents a connection to a database. The `Statement`, `PreparedStatement`, and `CallableStatement` interfaces support the execution of various kinds of SQL statements. `ResultSet` is a set of results returned by the database in response to a SQL query. The `ResultSetMetaData` interface provides metadata about a result set, while `DatabaseMetaData` provides metadata about the database as a whole.

The java.sql package contains API for the following:

- Making a connection with a data source

    - DriverManager class
    - Driver interface
    - DriverPropertyInfo class
    - Connection interface
- Custom mapping an SQL user-defined type to a class in the Java programming language
    - SQLData interface
    - SQLInput interface
    - SQLOutput interface
- Providing information about the database and the columns of a ResultSet object
    - DatabaseMetaData interface
    - ResultSetMetaData interface
- Throwing exceptions
    - SQLException thrown by most methods when there is a problem accessing data and by some methods for other reasons
    - SQLWarning thrown to indicate a warning
    - DataTruncation thrown to indicate that data may have been truncated
    - BatchUpdateException thrown to indicate that not all commands in a batch update executed successfully
- Providing security
    - SQLPermission interface

Metadata

RowSetMetaData: This interface, derived from the ResultSetMetaData interface, provides information about the columns in a RowSet object. An application can use RowSetMetaData methods to find out how many columns the rowset contains and what kind of data each column can contain. The RowSetMetaData interface provides methods for setting the information about columns, but an application would not normally use these methods. When an application calls the RowSet method execute, the RowSet object will contain a new set of rows, and its RowSetMetaData object will have been internally updated to contain information about the new columns.

The Reader/Writer Facility

A RowSet object that implements the RowSetInternal interface ca n call on the RowSetReader object associated with it to populate itself with data. It can also call on the RowSetWriter object associated with it to write any changes to its rows back to the data source from which it originally got the rows. A rowset that remains connected to its data source does not need to use a reader and writer because it can simply operate on the data source directly.

RowSetInternal: By implementing the RowSetInternal interface, a RowSet object gets access to its internal state and is able to call on its reader and writer. A rowset keeps track of the values in its current rows and of the values that immediately preceded the current ones, referred to as the *original* values. A rowset also keeps track of (1) the parameters that have been set for its command and (2) the connection that was passed to it, if any. A rowset uses the RowSetInternal methods behind the scenes to get access to this information. An application does not normally invoke these methods directly.

RowSetReader: A disconnected RowSet object that has implemented the RowSetInternal interface can call on its reader (the RowSetReader object associated with it) to populate it with data. When an application calls the RowSet.execute method, that method calls on the rowset's reader to do much of the work. Implementations can vary widely, but generally a reader makes a connection to the data source, reads data from the data source and populates the rowset with it, and closes the connection. A reader may also update the RowSetMetaData object for its rowset. The rowset's internal state is also updated, either by the reader or directly by the method RowSet.execute.

RowSetWriter: A disconnected RowSet object that has implemented the RowSetInternal interface can call on its writer (the RowSetWriter object associated with it) to write changes back to the underlying data source.

Implementations may vary widely, but generally, a writer will do the following:

- Make a connection to the data source
- Check to see whether there is a conflict, that is, whether a value that has been changed in the rowset has also been changed in the data source
- Write the new values to the data source if there is no conflict
- Close the connection

The RowSet interface may be implemented in any number of ways, and anyone may write an implementation. Developers are encouraged to use their imaginations in coming up with new ways to use rowsets

JDBC PROCESS

2.11.1 **JDBC Data structure**

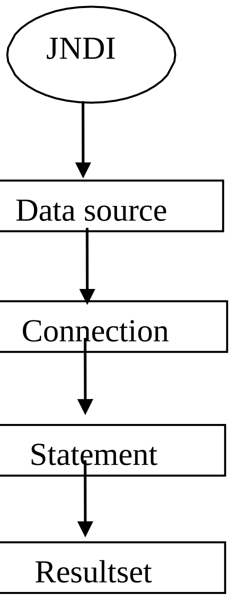


Figure 2.11.1 Data structure of JDBC

Steps involved in JDBC Process:

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

12.11.2 JDBC: Details of Process

1. Load the driver

```
try
{
Class.forName("connect.microsoft.MicrosoftDriver");
Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch { ClassNotFoundException cnfe)
{
System.out.println("Error loading driver: " cnfe);
}
```

2. Define the Connection URL

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host + ":" + port + ":" +_
dbName;
String sybaseURL = "jdbc:sybase:Tds:" + host +
":" + port + ":" +
"?SERVICENAME=" + dbName;
```

3. **Establish the Connection**

```
String username = "jay_debesee";
String password = "secret";
Connection connection =_
DriverManager.getConnection(oracleURL,username, password);
```

• Optionally, look up information about the database

```
DatabaseMetaData dbMetaData = connection.getMetaData();
String productName = dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion = dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);
```

4. Create a Statement

```
Statement statement = connection.createStatement();
```

5. Execute a Query

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

– To modify the database, use executeUpdate, supplying a string that uses UPDATE, INSERT, or DELETE
– Use setQueryTimeout to specify a maximum delay to wait for results

6. Process the Result

```
while(resultSet.next()) {
System.out.println(resultSet.getString(1) + " " +
resultSet.getString(2) + " " +
resultSet.getString(3));
}
```

First column has index 1, not 0
– ResultSet provides various get*Xxx* methods that take a colu index *or column name* and returns the data
– You can also access result meta data (column names, etc.)

7. Close the Connection

```
connection.close();
```

– Since opening a connection is expensive, postpone this step if additional database operations are expected

Statement Objects

Through the Statement object, SQL statements are sent to the database.

– Three types of statement objects are available:
• **Statement**
– For executing a simple SQL statement
• **PreparedStatement**
– For executing a precompiled SQL statement passing in parameters
• **CallableStatement**
– For executing a database stored procedure

Statement Methods

• executeQuery
  – Executes the SQL query and returns the data in a table (ResultSet)
  – The resulting table may be empty but never null

```
ResultSet results =
            statement.executeQuery("SELECT a, b FROM_ table");
```

• executeUpdate
  – Used to execute for INSERT, UPDATE, or DELETE, SQL statements
  – The return is the number of rows that were affected in the database
  – Supports Data Definition Language (DDL) statements

CREATE TABLE, DROP TABLE and ALTER TABLE
int rows = statement.executeUpdate("DELETE FROM EMPLOYEES" + _ "WHERE STATUS=0");

• **execute**
– Generic method for executing stored procedures and prepared statements
– Rarely used (for multiple return result sets)
– The statement execution may or may not return a ResultSet (use tatement.getResultSet). If the return value is true, two or more result sets were produced

• **getMaxRows/setMaxRows**
– Determines the maximum number of rows a ResultSet may contain
– Unless explicitly set, the number of rows is unlimited (return value of 0)

• **getQueryTimeout/setQueryTimeout**
  – Specifies the amount of a time a driver will wait for a statement to complete before throwing a SQLException

RESULTSET

ResultSet and Cursors

The rows that satisfy a particular query are called the result set. The number of rows returned in a result set can be zero or more. A user can access the data in a result set using a cursor one row at a time from top to bottom. A cursor can be thought of as a pointer to the rows of the result set that has the ability to keep track of which row is currently being accessed. The JDBC API supports a cursor to move both forward and backward and also allowing it to move to a specified row or to a row whose position is relative to another row.

Types of Result Sets

The ResultSet interface provides methods for retrieving and manipulating the results of executed queries, and ResultSet objects can have different functionality and characteristics. These characteristics are result set type, result set concurrency, and cursor hold ability.

The type of a ResultSet object determines the level of its functionality in two areas: the ways in which the cursor can be manipulated, and how concurrent changes made to the underlying data source are reflected by the ResultSet object.

The sensitivity of the ResultSet object is determined by one of three different ResultSet types:

TYPE_FORWARD_ONLY — the result set is not scrollable i.e. the cursor moves only forward, from before the first row to after the last row.

TYPE_SCROLL_INSENSITIVE — the result set is scrollable; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position.

TYPE_SCROLL_SENSITIVE — the result set is scrollable; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position.

Before you can take advantage of these features, however, you need to create a scrollable ResultSet object. The following line of code illustrates one way to create a scrollable ResultSet object:

```
Statement stmt =       con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
                       ResultSet srs = stmt.executeQuery(" .... ");
```

The first argument is one of three constants added to the ResultSet API to indicate the type of a ResultSet object: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE. The second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable: CONCUR_READ_ONLY and CONCUR_UPDATABLE. If you do not specify any constants for the type and updatability of a ResultSet object, you will automatically get one that is TYPE_FORWARD_ONLY and CONCUR_READ_ONLY.

Result Set Methods

When a ResultSet object is first created, the cursor is positioned before the first row. To move the cursor, you can use the following methods:

- ❖ next() - moves the cursor forward one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned after the last row.
- ❖ previous() - moves the cursor backwards one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned before the first row.
- ❖ first() - moves the cursor to the first row in the ResultSet object. Returns true if the cursor is now positioned on the first row and false if the ResultSet object does not contain any rows.
- ❖ last() - moves the cursor to the last row in the ResultSet object. Returns true if the cursor is now positioned on the last row and false if the ResultSet object does not contain any rows.
- ❖ beforeFirst() - positions the cursor at the start of the ResultSet object, before the first row. If the ResultSet object does not contain any rows, this method has no effect.
- ❖ afterLast() - positions the cursor at the end of the ResultSet object, after the last row. If the ResultSet object does not contain any rows, this method has no effect.
- ❖ relative(int rows) - moves the cursor relative to its current position.
- ❖ absolute(int n) - positions the cursor on the n-th row of the ResultSet object

SUMMARY

The JDBC API is a natural Java interface to the basic SQL abstractions and concepts. It builds on ODBC rather than starting from scratch, so programmers familiar with ODBC will find it very easy to learn JDBC. An API for database-independent connectivity between the J2EE platform and a wide range of data sources. The JDBC API supports both models for database access: two-tier (direct database access) and three-tier (communication with the database over a "middle-tier" on the database server or a separate machine). The purpose of the JDBC package is to provide vendor-neutral access to relational databases.

KEY TERMS

- JDBC API: support application to JDBC manager communications
- JDBC driver manager: support JDBC manager to driver implementation

- JDBC-ODBC bridge: provides JDBC access using ODBC as the communications pipe.
- Native API, partly Java Driver: converts JDBC calls directly into calls on the client to the native database API. This requires binary code on the client machine.
- JDBC Net Driver: translates JDBC into DBMS independent protocol. This independent protocol is translated by a server into native DBMS calls. Generally deployed using a middleware server.
- Native Protocol - Pure Java: converts JDBC into DBMS native network calls.

QUESTIONS AND EXERCISES

Short Answer Questions

1. Define Data, Database and Database Schema
2. Give a brief notes on different levels of Database Schema
3. What is JDBC API?
4. Write short notes on Statement objects
5. List two different packages used in JDBC API
6. List various Resultsets methods

Long –Answer Question

1. Explain the six steps to create a database schema with suitable examples
2. Write a brief note on Normalizing Data with suitable example
3. Explain the architecture of JDBC
4. Describe in Detail four different Types of JDBC Driver
5. Explain about Resultsets in Detail.
6. Write a database connectivity program in j2EE for maintaining students detail

References

1. Jim Keogh , ―The Complete Reference J2SE‖, 1st Edition, Tata McGraw Hill Edition, New Delhi:2002.
2. Deepak Vohra , JDBC 4.0 and Oracle JDeveloper for J2EE Development
3. Derek Ashmore , The J2EE Architect's Handbook.
4. JDBC - http://java.sun.com/docs/books/tutorial/jdbc

UNIT-II
J2EE

S. No.	QUESTIONS	OPTION 1	OPTION 1	OPTION 1	OPTION 1	ANSWER
1.	A _________ is a collection of data.	field	record	database	DBMS	database
2.	A database is managed by ___________.	SQL	DBMS	JAVA	J2EE	DBMS
3.	___________ refers to an atomic unit.	field	data	record	DBMS	data
4.	A ___________ is a component of a database that contains data in the form of rows and columns.	tuple	table	record	attribute	table
5.	A __________ is a document that defines all components of a database.	SQL	database schema	table	file	database schema
6.	The best way to identify attributes of an entity is by analyzing _________ of the entity.	instances	fields	data	records	instances
7.	The __________ describes the number of characters used to store values of the attribute.	attribute range	attribute size	attribute format	attribute type	attribute size
8.	The _________ uniquely identifies the attribute from other attributes of the same entity.	attribute name	attribute size	attribute format	attribute type	attribute name

9.	A __________ is nearly identical to the data type of a column in a table.	attribute name	attribute size	attribute format	attribute type	attribute type
10.	The __________ is minimum and maximum values that can be assigned to an attribute.	attribute name	attribute size	attribute format	attribute range	attribute range
11.	The __________ is the value that is automatically assigned to the attribute.	attribute name	attribute size	attribute definition value	attribute type	attribute definition value
12.	The _________ consists of the way in which an attribute appears in the existing system.	attribute format	attribute size	attribute definition value	attribute type	attribute format
13.	The __________ identifies the origin of the attribute value.	attribute format	attribute source	attribute definition value	attribute type	attribute source
14.	A __________ is free form text that is used to describe an attribute.	acceptable values	required values	comments	attribute values	comments
15.	___________ must be reduced to data elements.	values	attributes	comments	information	attributes
16.	The unique name given to the data element is called ___________.	data name	data type	data size	attribute	data name
17.	A ___________ describes the kind of values that are associated with the data.	data name	data type	data size	attribute	data type
18.	The __________ is the maximum number of characters required to represent values of the data.	data name	data type	data size	attribute	data size
19.	The nature of the data provide a hint to the __________.	data name	data type	data size	attribute	data name
20.	___________ should have as few characters as possible to identify the data.	data name	data type	data size	attribute	data name

21.	A ___________ can be abbreviated using components of the name.	data name	data type	data size	attribute	data name
22.	A ___________ describes the characteristics of data associated with a data element.	data name	data type	data size	attribute	data type
23.	___________ data stores alphabetical characters and punctuations.	Character	Alpha	Alphanumeric	Numeric	Character
24.	__________ data stores only alphabetical characters.	Character	Alpha	Alphanumeric	Numeric	Alpha
25.	___________ data stores alphabetical characters, punctuations, and numbers.	Character	Alpha	Alphanumeric	Numeric	Alphanumeric
26.	___________ data stores numbers only.	Character	Alpha	Alphanumeric	Numeric	Numeric
27.	_________ data stores date and time values.	Character	Alpha	Date/Time	Numeric	Date/Time
28.	___________ data stores one of two values – yes or no.	Character	Alpha	Alphanumeric	Logical	Logical
29.	___________ data stores large text fields, images, and other binary data.	Character	Alpha	Alphanumeric	Large Object	Alphanumeric
30.	_________ is the process of organizing data elements into related groups to minimize redundant data and to assure data integrity.	Transaction	Normalization	Grouping	Creation	Normalization
31.	There are __________ normal forms.	2	3	4	5	5
32.	A common way to organize data elements into ________ is to first assemble a list of all data elements.	groups	text	objects	None of the above	groups

33.	A ________ requires the information to be atomic.	1 NF	2 NF	3 NF	4 NF	1 NF
34.	The _________ requires the data to be in the first normal form.	1 NF	2 NF	3 NF	4 NF	2 NF
35.	The _________ requires that data elements to be in second normal form.	1 NF	2 NF	3 NF	4 NF	3 NF
36.	A _________ is a data element that uniquely identifies a row of data elements within a group.	primary key	secondary key	tertiary key	foreign key	primary key
37.	A ________ occurs when data depends on other data such as when nonkey data is dependent on a primary key.	redundancy	normalization	functional dependency	transitive dependency	functional dependency
38.	A _______ is a functional dependency between two or more nonkey data elements.	redundancy	normalization	functional dependency	transitive dependency	transitive dependency
39.	A _________ is a primary key of another group used to draw a relationship between two groups of data elements.	primary key	secondary key	tertiary key	foreign key	foreign key
40.	The relationship between primary keys and foreign keys of data groups is called _________.	functional dependency	referential integrity	transitive dependency	None of the above	referential integrity
41.	There are ________ types of JDBC drivers.	2	3	4	5	4
42.	The JDBC process is divided into _________ routines.	2	3	4	5	5
43.	The __________ method is used to load the JDBC driver.	Class.forName()	Results.next()	System.out.println()	DB.createStatement()	Class.forName()

44.	The ________ method returns a connection interface that is used throughout the process to reference the database.	Class.forName()	Results.next()	DriverManager.getConnection()	DB.createStatement()	DriverManager.getConnection()
45.	The ________ method is used to create a statement object.	Class.forName()	Results.next()	DriverManager.getConnection()	Connect.createStatement()	Connect.createStatement()
46.	The ________ method is called to terminate the statement.arameter.	Class.forName()	Results.next()	Db.close()	Connect.createStatement()	Db.close()
47.	The ________ method of the ResultSet object is used to copy the value of a specified column in the current row of the ResultSet to a string object.	Class.forName()	Results.next()	Db.close()	getString()	getString()
48.	The URL consists of ________ parts.	2	3	4	5	3
49.	The statement object contains the method, which is passed the query as an argument.	Results.next()	Class.forName()	executeQuery()	Db.createStatement()	executeQuery()
50.	The _________ method of the statement object is used when there may be multiple results returned.	Results.next()	Class.forName()	executeQuery()	execute()	execute()
51.	The _________ method of the connection object is called to return a statement object.	createStatement()	Class.forName()	executeQuery()	execute()	createStatement()
52.	The _________ method of the connection object is called to return the PreparedStatement object	createStatement()	Class.forName()	executeQuery()	preparedStatement()	preparedStatement()
53.	The ________ object is used to call a stored procedure from within a J2EE object.	statement	preparedstatement	callableStatement	ResultSet	callableStatement

54.	The CallableStatement object used _________ types of parameter when calling a stored procedure.	2	3	4	5	3
55.	The _________ parameter contains any data that needs to be passed to the stored procedure.processed by the CPU?	IN	OUT	INOUT	None of the above	IN
56.	The __________ object is used whenever a J2EE component needs to immediately execute a query without first having the query compiled.	statement	preparedstatement	callableStatement	ResultSet	statement
57.	A SQL query can be preempted and executed using the __________ object.	statement	preparedstatement	callableStatement	ResultSet	preparedstatement
58.	The ________ object contains methods that are used to copy data from the ResultSet into a java collection object or variable for further processing.	statement	preparedstatement	callableStatement	ResultSet	ResultSet
59.	The __________ parameter is a single parameter that is used to both pass information to the stored procedure and retrieve information from a stored procedure.	IN	OUT	INOUT	None of the above	INOUT
60.	The _________ parameter contains a value returned by the stored procedures. The future generation of computers?	IN	OUT	INOUT	None of the above	OUT
61.	The ________ mthod returns a Boolean true if the row contains data.	getString()	next()	execute()	d) close()	next()

62.	There are _______ methods of the ResultSet object that are used to position the virtual cursor.	3	4	5	6	6
63.	The _________ method moves the virtual cursor specified to the first row in the ResultSet.	first()	next()	relative()	absolute()	first()
64.	The _________ method moves the virtual cursor the specified number of rows contained in the parameter.	first()	next()	relative()	absolute()	relative()
65.	A value in a column of the ResultSet can be replaced with a NULL value by using the _________ method.	updataeNull()	updateRow()	updateString()	deleteRow()	updataeNull()
66.	The ________ method is used to change the value of the column of the ResultSet.	updataeNull()	updateRow()	updateString()	deleteRow()	updateString()
67.	The _________ method is used to remove a row from a ResultSet.	updataeNull()	updateRow()	updateString()	deleteRow()	deleteRow()
68.	The _______ method returns the product name of the database.	getDatabaseProductName()	getUserName()	getURL()	getschemas()	getDatabaseProductName()
69.	The ________ method returns the username.	getDatabaseProductName()	getUserName()	getURL()	getschemas()	getUserName()

70.	The _________ method returns the URL of the database.	getDatabaseProductName()	getUserName()	getURL()	getschemas()	getURL()
71.	The _________ method returns all the schema names available in the database.	getDatabaseProductName()	getUserName()	getURL()	getschemas()	getschemas()
72.	The __________ method returns primary keys.	getPrimaryKeys()	getProcedures()	getTables()	getColumnCount()	getPrimaryKeys()
73.	The ________ method returns stored procedure names.	getPrimaryKeys()	getProcedures()	getTables()	getColumnCount()	getProcedures()
74.	The _________ method returns names of table in the database.	getPrimaryKeys()	getProcedures()	getTables()	getColumnCount()	getTables()
75.	The __________ method returns the number of columns contained in the ResultSet.	getPrimaryKeys()	getProcedures()	getTables()	getColumnCount()	getColumnCount()

UNIT 3
JAVA SERVLETS

OVERVIEW OF SERVLET

Servlets are Java programming language objects that dynamically process requests and construct responses. The **Java Servlet API** allows a software developer to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets are the Java counterpart to non-Java dynamic Web content technologies such as PHP, CGI and ASP.NET, and as such some find it easier to think of them as 'Java scripts'. Servlets can maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting.

The servlet API, contained in the Java package hierarchy javax.servlet, defines the expected interactions of a Web container and a servlet. A Web container is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

A Servlet is an object that receives a request and generates a response based on that request. The basic servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package javax.servlet.http defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the Web server and a client. Servlets may be packaged in a WAR file as a Web application.

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.

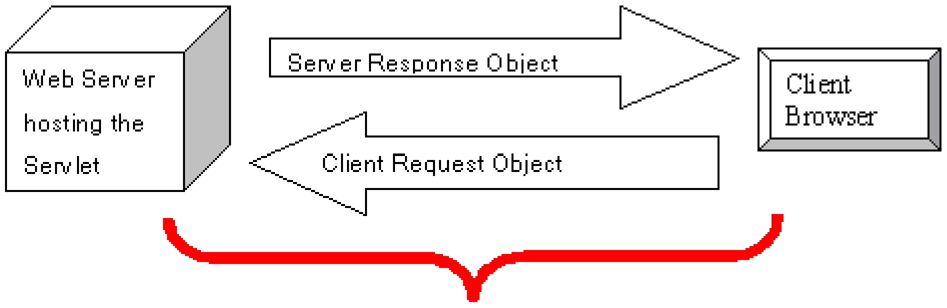


Figure 3.1.1: HTTP request response model.

Web based Client/server interaction uses the HTTP (hypertext transfer protocol). HTTP is a stateless protocol based on a request and response model with a small, finite number of request methods like GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE, CONNECT, etc. The response contains the status of the response and meta information describing the response. Most of the servlet-based web applications are built around the framework of the HTTP request/response model (Figure 3.1.1).

CGI VERSUS SERVLET

When a CGI program (or script) is invoked what typically happens is that a new process is spawned to handle the request. This process is external to that of the web server and as such you have the overhead of creating a new process and context switching etc. If you have many requests for a CGI script then you can imagine the consequences! Of course this is a generalization and there are wrappers for CGI that allow them to run in the same process space as the web server. Java Servlets on the other hand actually run inside the web server (or Servlet engine).

The developer writes the Servlet classes compiles them and places them somewhere that the server can locate them. The first time a Servlet is requested it is loaded into memory and cached. From then on the same Servlet instance is used with different requests being handled by different threads. The below table 3.3.1 depicts the difference between CGI and Servlet

Table 3.3.1 Difference Between CGI And Servlet

CGI	Servlet
Written in C, C++, Visual Basic and Perl	Written in Java
Difficult to maintain, non-scalable, non-manage	Powerful, reliable, and efficient
Prone to security problems of programming language	Improves scalability, reusability (component based)
Resource Intensive and inefficient	Leverages Built-in security of Java programming language
Platform and application-specific	Platform independent and portable

BENEFITS OF JAVA SERVLETS

When developing server-side software applications, its size becomes larger and automatically complexity intrudes in. It is always helpful if such a large application gets broken into discreet modules that are each responsible for a specific task. This divide and conquer principle helps to maintain and understand easily. Java Servlets provide a way to modularize user application.

Advantages of Servlets

1. No CGI limitations

2. Abundant third-party tools and Web servers supporting Servlet
3. Access to entire family of Java APIs
4. Reliable, better performance and scalability
5. Platform and server independent Secure
6. Most servers allow automatic reloading of Servlet's by administrative action.

SERVLET REQUEST AND RESPONSE

There are three different players in figure 3.4.1. They are browser, web server, and servlet container. In many cases, a web server and a servlet container are running in a same machine even in a single virtual machine. So they are not really distinguished in many cases. The role of the web server is to receive HTTP request and then passes it to the web container or servlet container which then creates Java objects that represent –HTTP request‖ and a –session‖ and then dispatches the request to the servlet by invoking service() method defined in the servlet

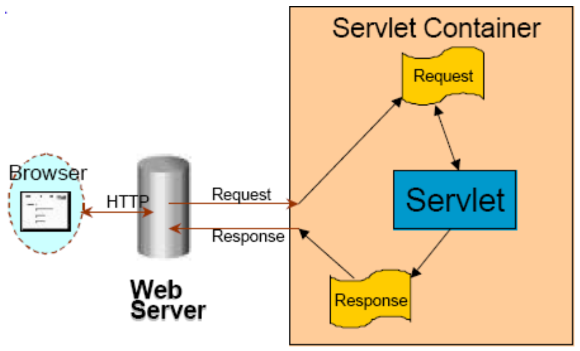


Figure 3.4.1 Three different players

And once the servlet handles the request, it creates a HTTP response, which is then sent to the client through the web server.

- HTTPServletRequest object
  - Information about an HTTP request
    - Headers
    - Query String
    - Session
    - Cookies
- HTTPServletResponse object
  - Used for formatting an HTTP response
    - Headers
    - Status codes
    - Cookies

SERVLET CLASSES AND INTERFACES

Servlet Request Interface

public interface **ServletRequest:** Defines an object to provide client request information to a servlet. The servlet container creates a ServletRequest object and passes it as an argument to the servlet's service method. A ServletRequest object provides data including parameter name and values, attributes, and an input stream. Interfaces that extend

ServletRequest can provide additional protocol specific data (for example, HTTP data is provided by **HttpServletRequest**)

ServletResponse Interface

public interface **ServletResponse:** Defines an object to assist a servlet in sending a response to the client. The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method

A subclass of HttpServlet must override at least one method, usually one of these:
• doGet, if the servlet supports HTTP GET requests
• doPost, for HTTP POST requests
• doPut, for HTTP PUT requests
• doDelete, for HTTP DELETE requests
• init and destroy, to manage resources that are held for the life of the servlet
Web clients usually activate a servlet in one of two ways:
• Get – Sends data as part of a URL
http://rmyers.com/servlet/Hello?name="john"
• Post – Sends data down the data stream following the request

Java Servlet Anatomy and Life Cycle

Anatomy of Java Servlets:
init()
– Invoked once when the servlet is first instantiated
– Perform any set-up in this method and Setting up a database connection
destroy()
– Invoked before servlet instance is removed.
-Perform any clean-up and Closing a previously created database connection

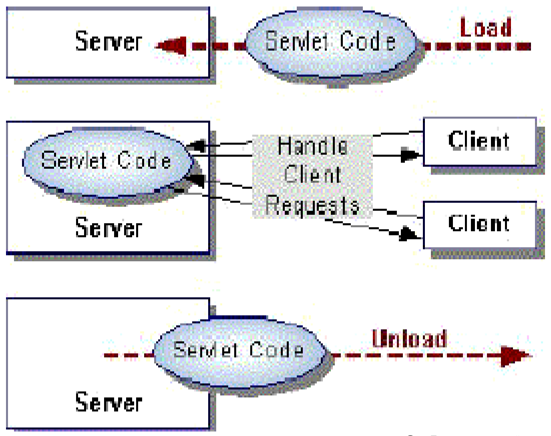


Figure 3.6.1.Function of doGet()

doGet()
– the doGet() function is called when the servlet is called via an HTTP GET
doPost()
– the doPost() function is called when the servlet is called via an HTTP POST. POSTs are a good way to get input from HTML forms

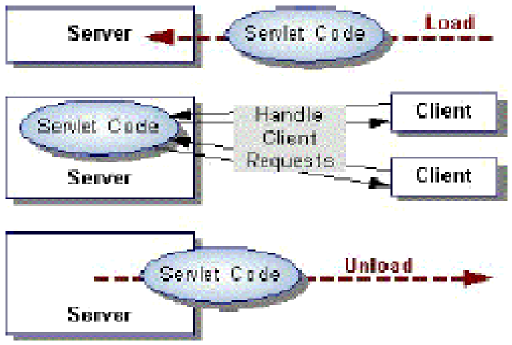


Figure 3.6.2 Function of doPost()

Life Cycle of Java Servlets:

The life cycle of a servlet is controlled by servlet-container in which the servlet has been deployed. When a HTTP request is mapped to a servlet, the container performs the following steps.

- ❖ If an instance of the servlet does not exist, the Web container
  - o Loads the servlet class
  - o Creates an instance of the servlet class
  - o Initializes the servlet instance by calling the init() method
- ❖ Invokes the service method, passing HttpServletRequest and HttpServletResponse objects as parameters.

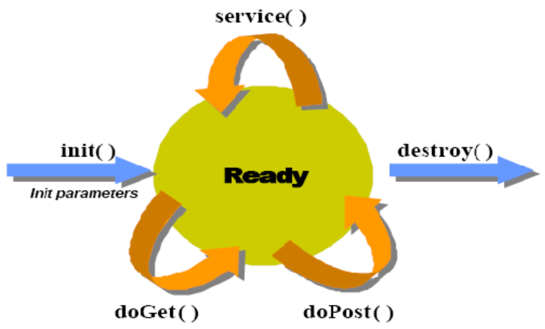


Figure 3.6.3 Methods used in Java Servlets

The init() method gets called once when a servlet instance is created for the first time. And then service() method gets called every time there comes a new request. Now service() method in turn calls doGet() or doPost() methods for incoming HTTP requests. And finally when the servlet instance gets removed, the destroy() method gets called. So init() and destroy() methods get called only once while service(),

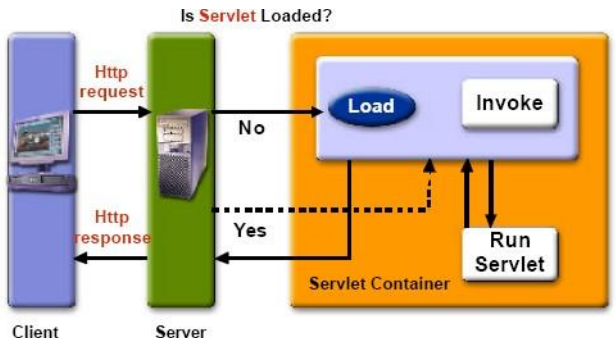


Figure 3.6.4 Httprequest and Httpresponse

doGet(), and doPost() methods are called a number of times depending on how many HTTP requests are received. As it was mentioned before, init () and destroy () methods are called only once, init() at the time service instance is created while destroy() gets called at the time servlet instance gets removed. And init() can be used to perform some set up operation such as setting up a database
connection and destroy() method is used to perform any clean up, for example, removing a previously created database connection.

Example for init():

```
public class CatalogServlet extends HttpServlet {
private BookDB bookDB;
// Perform any one-time operation for the servlet,
// like getting database connection object.
// Note: In this example, database connection object is assumed
// to be created via other means (via life cycle event mechanism)
// and saved in ServletContext object. This is to share a same
// database connection object among multiple servlets.
public void init() throws ServletException {
bookDB = (BookDB)getServletContext().
getAttribute("bookDB");
if (bookDB == null) throw new
UnavailableException("Couldn't get database.");
}
...
}
```

Example: destroy()

```
public class CatalogServlet extends HttpServlet {
private BookDB bookDB;
public void init() throws ServletException {
bookDB = (BookDB)getServletContext().
getAttribute("bookDB");
if (bookDB == null) throw new
UnavailableException("Couldn't get database.");
}
public void destroy() {
bookDB = null;
```

}
…
This is destroy example code again from CatalogServlet code. Here destroy() method nulling the local variable that contains the reference to database connection.
service() methods take generic requests and responses:

- service(ServletRequest request, ServletResponse response)
- doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponse response)

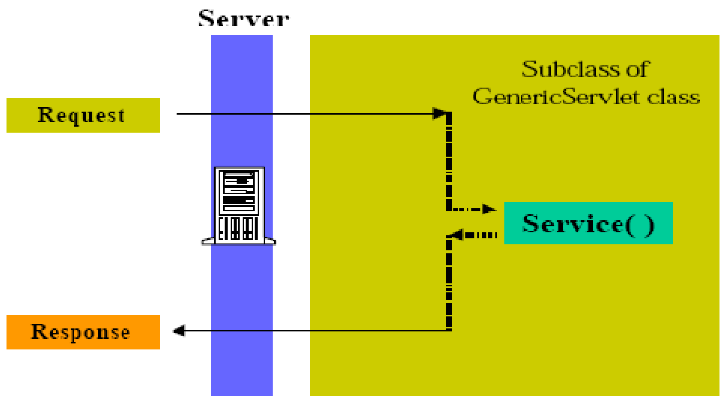


Figure 3.6.5 using service() method to invoke GenericServlet class

This Figure 3.6.5 shows how service () method of a subclass of GenericServlet class is invoked.

doGet() and doPost() Methods

Using doGet() and doPost() it is possible to do the following functions:

- Can able to extract client sent information such as user-entered parameter values that were sent as query string.
- To set and get attributes to and from scope objects.
- Perform some business logic or access the database.
- Optionally include or forward your requests to other web components.
- Populate HTTP response message and then send it to client.

Example: Simple doGet()

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
Public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Just send back a simple HTTP response
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<title>First Servlet</title>");
out.println("<big>Hello J2EE Programmers! </big>");
}
}
```

This is a very simple example code of doGet() method. In this example, a simple HTTP response message is created and then sent back to client

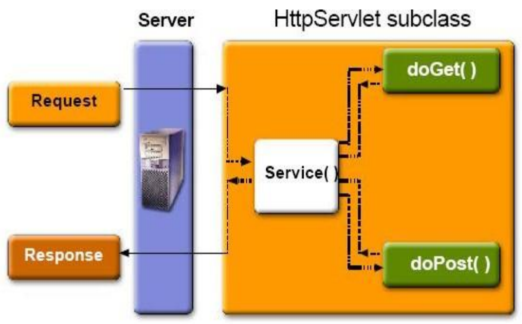


Figure 3.6.6 HttpServlet subclass

READING DATA FROM A CLIENT

A Client uses either the GET or POST Method to pass information to a java servlet. The doGet() or doPost() netgid us called in the Java Servlet depending on the method used by the client.

Data sent by a client is read into java servlet by calling the getParameters() method of the HttpservletRequest object that instantiated in the argument list of the doGet() and dopost() methods. The getParameters() method requires one argument, which is the name of the parameter that contains the data sent by the client. The getParameters() method returns a String object. The String object contains the value of the parameter, if the client assigns a value to the parameter. An empty string object is returned if the client didn't assign a value to the parameter. Also, a null is returned if the parameter isn't received from the client.

A HTML form can contain a set of check boxes or other form objects that have the same data name but different values. This means that data received from a client might have multiple occurrences of the same parameter name.

The user can read a set of parameters that have the same name by calling the getParameterValues() method. The getParameterValues() method has one argument which is the name of the parameter, and returns an array of string objects. Each element of the array contains a value of the set of parameters. The getParameterValues( ) method returns a null if data received from the client doesn't contain the parameter named in the argument.

User can retrieve all the parameters by calling the getParameterNames() method. The getParameterNames() method does not require an argument and returns an Enumeration. Parameter names appear in any order and can be cast to String object and used with the getParameter() and getParameterValues() methods.

Figure conatins an HTML form that prompts a user to enter their name , when the user selects the Submit button, the browser calls the URL /servlet/HelloServlet Java Servlet and sends the username as data. Figure illustrates the HelloServlet.class Java Servlet that reads data sent by this form. In this example the getParameter() method returns a string that is assigned to the email String object called email. The value of the email String object is then returned to the browser in the form of an HTML page.

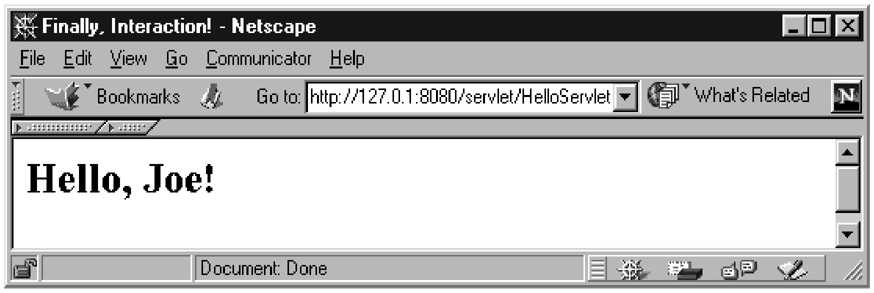
```
<HTML>
<HEAD><TITLE>Greetings Form</TITLE></HEAD>
<BODY>
<FORM METHOD=GET ACTION="/servlet/HelloServlet">
What is your name?
<INPUT TYPE=TEXT NAME=username SIZE=20>
<INPUT TYPE=SUBMIT VALUE="Introduce Yourself">
</FORM>
</BODY>
</HTML>
```

This form submits a form variable named username to the URL /servlet/HelloServlet.

The HelloServlet itself does little more than create an output stream, read the username form variable, and print a nice greeting for the user.
Here's the code:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
String name;
name= req.getParameter("username");
resp.setContentType("text/html");
PrintWriter out = resp.getWriter();
out.println("<HTML>");
out.println("<HEAD><TITLE>Finally, interaction!</TITLE></HEAD>");
out.println("<BODY><H1>Hello, " + name+"!</H1>");
out.println("</BODY></HTML>");
}
}
```

Result:


Finally, Interaction! - Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://127.0.1:8080/servlet/HelloServlet
What's Related
Document: Done

Hello, Joe!

READING HTTP REQUEST HEADERS

When an HTTP client (e.g. a browser) sends a request, it is required to supply a request line (usually GET or POST). If it wants to, it can also send a number of headers, all of which are optional except for Content-Length, which is required only for POST requests. Here are the most common headers:

- Accept The MIME types the browser prefers.
- Accept-Charset The character set the browser expects.
- Accept-Encoding The types of data encodings (such as gzip) the browser knows how to decode. Servlets can explicitly check for gzip support and return gzipped HTML pages to browsers that support them, setting the Content-Encoding response header to indicate that they are gzipped. In many cases, this can reduce page download times by a factor of five or ten.
- Accept-Language The language the browser is expecting, in case the server has versions in more than one language.
- Authorization Authorization info, usually in response to a WWW-Authenticate header from the server
- Connection Use persistent connection? If a servlet gets a Keep-Alive value here, or gets a request line indicating HTTP 1.1 (where persistent connections are the default), it may be able to take advantage of persistent connections, saving significant time for Web pages that include several small pieces (images or applet classes). To do this, it needs to send a Content-Length header in the response, which is most easily accomplished by writing into a ByteArrayOutputStream, then looking up the size just before writing it out.
- Content-Length (for POST messages, how much data is attached)
- Cookie (one of the most important headers; see separate section in this tutorial on handling cookies)
- From (email address of requester; only used by Web spiders and other custom clients, not by browsers)
- Host (host and port as listed in the original URL)
- If-Modified-Since (only return documents newer than this, otherwise send a 304 Not Modified" response)
- Pragma (the no-cache value indicates that the server should return a fresh document, even if it is a proxy with a local copy)
- Referer (the URL of the page containing the link the user followed to get to current page)
- User-Agent (type of browser, useful if servlet is returning browser-specific content)

UA-Pixels, UA-Color, UA-OS, UA-CPU (nonstandard headers sent by some Internet Explorer versions, indicating screen size, color depth, operating system, and cpu type used by the browser's system)

SENDING DATA TO A CLIENT

A java Servlet responds to a client request by reading client data and the HTTP request headers, then processing information based on the nature of the request. For example, a client request for information about merchandise in an online product catalog requires the Java Servlet to search the product database to retrieve product information and then format the product information into a web page which is returned to the client.

There are two ways in which a java Servlet replied to a client request. These are by sending information to the response stream and by sending information in the HTTP response header.

The HTTP response header is similar to the HTTP request header except the contents of the HTTP response header are generated by the web server that responds to the client's request. Information is sent to the response stream by creating an instance of the PrintWriter object and then using the println() method to transmit the information to the client.

An Http response header contains a status line, response headers, and a blank line, followed by the document. There are three components to the status line these are the HTTP version number, a status code and a brief message associated with the status code. example :

```
HTTP/1.1 200 OK
Content-type : text/plain
My response
```

In the above example The HTTP Version number is 1.1 and the status code is 200, indicating that everything is fine with the request that was received from the client. OK is the message that is associated with the status code. This example contains HTTP response Header, which is Content-Type that identifies the document Mime type as plain text. The document contains the expression My response.

WORKING WITH COOKIES

A cookie is a bit of information sent by a web server to a browser that can later be read back from that browser. When a browser receives a cookie, it saves the cookie and thereafter sends the cookie back to the server each time it accesses a page on that server, subject to certain rules. Because a cookie's value can uniquely identify a client, cookies are often used for session tracking. Version 2.0 of the Servlet API provides the `javax.servlet.http.Cookie` class for working with cookies. The HTTP header details for the cookies are handled by the Servlet API.

Create a cookie with the `Cookie()` constructor:

```
public Cookie(String name, String value)
```

This creates a new cookie with an initial name and value. The rules for valid names and values are given in Netscape's Cookie Specification and RFC 2109.

A servlet can send a cookie to the client by passing a `Cookie` object to the `addCookie()` method of `HttpServletResponse`:

```
public void HttpServletResponse.addCookie(Cookie cookie)
```

This method adds the specified cookie to the response. Additional cookies can be added with subsequent calls to `addCookie()`. Because cookies are sent using HTTP headers, they should be added to the response before you send any content. Browsers are only required to accept 20 cookies per site, 300 total per user, and they can limit each cookie's size to 4096 bytes.

The code to set a cookie looks like this:

```
Cookie cookie = new Cookie("ID", "123");
res.addCookie(cookie);
```

A servlet retrieves cookies by calling the `getCookies()` method of `HttpServlet-Request`:

public Cookie[] HttpServletRequest.getCookies()
This method returns an array of `Cookie` objects that contains all the cookies sent by the browser as part of the request or `null` if no cookies were sent.
The code to fetch cookies looks like this:

```
Cookie[] cookies = req.getCookies();
if (cookies != null) {
  for (int i = 0; i < cookies.length; i++) {
    String name = cookies[i].getName();
    String value = cookies[i].getValue();
  }
}
```

The following methods are used to set these attributes:

- **public void Cookie.setVersion(int v) :** Sets the version of a cookie. Servlets can send and receive cookies formatted to match either Netscape persistent cookies (Version 0) or the newer, somewhat experimental, RFC 2109 cookies (Version 1). Newly constructed cookies default to Version to maximize interoperability.
- **public void Cookie.setDomain(String pattern):** Specifies a domain restriction pattern. A domain pattern specifies the servers that should see a cookie. By default, cookies are returned only to the host that saved them. Specifying a domain name pattern overrides this. The pattern must begin with a dot and must contain at least two dots. A pattern matches only one entry beyond the initial dot. For example, ".foo.com" is valid and matches www.foo.com and upload.foo.combut not www.upload.foo.com. For details on domain patterns, see Netscape's Cookie Specification and RFC 2109.
- **public void Cookie.setMaxAge(int expiry):**Specifies the maximum age of the cookie in seconds before it expires. A negative value indicates the default, that the cookie should expire when the browser exits. A zero value tells the browser to delete the cookie immediately.
- **public void Cookie.setPath(String uri):** Specifies a path for the cookie, which is the subset of URIs to which a cookie should be sent. By default, cookies are sent to the page that set the cookie and to all the pages in that directory or under that directory. For example, if /servlet/CookieMonster sets a cookie, the default path is "/servlet". That path indicates the cookie should be sent to /servlet/Elmo and to /servlet/subdir/BigBird--but not to the /Oscar.html servlet alias or to any CGI programs under /cgi-bin. A path set to "/" causes a cookie to be sent to all the pages on a server.

SUMMARY

After going through this unit you will understand the role of Servlet in big picture of J2EE. **A**S soon as the Web began to be used for delivering services, service providers recognized the need for dynamic content. Applets, one of the earliest attempts toward this goal, focused on using the client platform to deliver dynamic user experiences. At the same time, developers also investigated using the server platform for this purpose. Initially, Common Gateway Interface (CGI) scripts were the main technology used to

generate dynamic content. Though widely used, CGI scripting technology has a number of shortcomings, including platform dependence and lack of scalability. To address these limitations, Java Servlet technology was created as a portable way to provide dynamic, user-oriented content.
Servlet request & response model. Servlet life cycle. Servlet scope objects. Servlet request and response: Status, Header, Body and Error Handling. Servlet from the standpoint of J2EE architecture, that is, what role Servlet plays in a multi-tier web-based application. Servlet is basically a web technology in which HTTP request is being received and handled and then proper HTTP response is being created and then returned to the client

KEY TERMS

- **Java Servlets :** Servlets are the Java platform technology of choice for extending and enhancing web servers
- **ServletRequest:** Defines an object to provide client request information to a servlet.
- **ServletResponse:** Defines an object to assist a servlet in sending a response to the client
- **init()** – Invoked once when the servlet is first instantiated
- **service ()** - This method gets called every time there comes a new request.
- **destroy()** – Invoked before servlet instance is removed
- **Http response header**: It contains a status line, response headers, and a blank line, followed by the document
- **Cookie** : It is a bit of information sent by a web server to a browser that can later be read back from that browser

QUESTIONS AND EXERCISES

Short Answer Questions

1. Write short notes on Java Servlet
2. Differentiate CGI and Java Servlet
3. List the advantage of Java Servlet
4. Give a brief notes on Java Servlet life cycle
5. Write a simple program using Java Servlet to display "Hello" message
6. Define any three HTTP1.1 Status Codes
7. Define Cookies
8. What is Tracking Session?

Long –Answer Question

1. Explain the anatomy of Java Servlet in detail
2. Describe in about Servlet request and response with suitable example
3. Explain about Servlet Classes and Interfaces
4. Write a java Servlet program to reading data from a client
5. Write program using java Servlet to send data to a client

Reference

1. Jim Keogh , ―The Complete Reference J2SE‖, 1st Edition, Tata McGraw Hill Edition, New Delhi:2002.
2. Deepak Vohra , JDBC 4.0 and Oracle JDeveloper for J2EE Development
3. Derek Ashmore , The J2EE Architect's Handbook
4. Servlets - http://java.sun.com/docs/books/tutorial/servlets

UNIT-III
J2EE

S. No.	Option 1	Option 2	Option 3	Option 4	Option 5	Answer
1.	There are ________ types of data.	2	3	4	5	2
2.	____________ is information received from the client that is typically either entered by the user into the user interface or generated by the user interface itself databases is called	Explicit data	Implicit data	CGI	Browser	Explicit data
3.	____________ is HTTP information that is generated by the client rather than the user.	Explicit data	Implicit data	CGI	Browser	Implicit data
4.	The result of processing a request is returned to the client as ____________.	Explicit data	Implicit data	CGI	Browser	Explicit data
5.	A ___________ is a server side program.	servlet	JSP	EJB	Java	servlet
6.	Java servlet remains alive after the request is fulfilled. This is called ___________.	persistence	reliability	Integrity	robustness	persistence
7.	A ___________ is a java class that reads requests sent from a client and responds by the sending information to the client.	servlet	JSP	EJB	EIS	servlet
8.	The doGet() method requires __________ arguments.	2	3	4	5	2

9.	The doPost() method requires ___________ arguments.	2	3	4	5	2
10.	Incoming data includes ___________ data.	implicit	explicit	implicity and explicit	None of the above	implicity and explicit
11.	The __________ method is used in conjunction with a PrintWriter to send outgoing explicit data such as text that appears on a webpage.	println()	setContentType()	doGet()	d) doPost()	println()
12.	The ___________ method is used to set the value for the ContentType HTTP header information.	println()	setContentType()	doGet()	d) doPost()	setContentType()
13.	The __________ method is called automatically when the java servlet is created.	init()	setContentType()	doGet()	doPost()	init()
14.	The ___________ method is called whenever a request for the java servlet is made to the web server.	init()	service()	doGet()	doPost()	service()
15.	The __________ method is called when an instance of a java servlet is removed from memory.	init()	service()	destroy()	doPost()	destroy()
16.	The ___________ method is not called when an abnormal occurrence such as a system malfunction causes the java servlet to abruptly terminate.	init()	service()	destroy()	doPost()	destroy()
17.	The web-app element should contain a servlet element with ___________ subelements.	2	3	4	5	3
18.	The _____________ contains the name used to access the java servlet.	servlet-name	servlet-class	init-param	None of the above	servlet-name

19.	A client uses the __________ method to pass information to a java servlet.	GET only	POST only	either GET or POST	None of the above	either GET or POST
20.	Data sent by a client is read into a java servlet by calling the __________ method.	getParameter()	doGet()	doPost()	getParameter Values()	getParameter()
21.	The _________ method returns a null if data received from the client doesnot contain the parameter named in the argument.	getParameter()	doGet()	doPost()	getParameter Values()	getParameterValue s()
22.	The _________ method does not require an argument and returns an enumeration.	getParameter()	getParameterN ames()	doPost()	getParameter Values()	getParameterName s()
23.	A request from a client contains __________ components.	2	3	4	5	2
24.	The HTTP Request Header ________ identifies the MIME type of data that can be handled by the browser that made the request.	Accept	Accept_Charse t	Accept_Language	Authorizatio n	Accept
25.	The HTTP Request Header __________ identifies the character sets that can be used by the browser that made the request.	Accept	Accept_Charse t	Accept_Language	Authorizatio n	Accept_Charset
26.	The HTTP Request Header __________ specifies the preferred languages that are used by the browser.	Accept	Accept_Charse t	Accept_Language	Authorizatio n	Accept_Language

27.	The HTTP Request Header __________ is used by a browser to identify the client to the java servlet whenever a protected web page is being processed.	Accept	Accept_Charset	Accept_Language	Authorization	Authorization
28.	The HTTP Request Header __________ identifies whether a browser can retrieve multiple files using the same socket, which is referred to as persistence.	Connection	Content-length	Cookie	Host	Connection
29.	The HTTP Request Header __________ contains the size of the data in bytes that are transmitted using the POST method.	Connection	Content-length	Cookie	Host	Content-length
30.	The HTTP Request Header __________ contains the host and port of the original URL	Connection	Content-length	Cookie	Host	Host
31.	The HTTP Request Header ________ signifies that the browser's requests should be fulfilled only if the data has changed since a specified date.	If-Modified-Since	If-Unmodified-Since	Referer	User-Agent	If-Modified-Since
32.	The HTTP Request Header __________ signifies that the browser's requests should be fulfilled only if the data is older than a specified date.	If-Modified-Since	If-Unmodified-Since	Referer	User-Agent	If-Unmodified-Since

33.	The HTTP Request Header ____________ contains the URL of the web page that is currently displayed in the browser.	If-Modified-Since	If-Unmodified-Since	Referer	User-Agent	Referer
34.	The HTTP Request Header __________ identifies the browser that made the request.	If-Modified-Since	If-Unmodified-Since	Referer	User-Agent	User-Agent
35.	HTTP __________ version uses the Keep-Alive message to keep a connection open.	1.1	1.2.	1.3	1.4	1.1
36.	There are ___________ ways in which a java servlet replies to a client request.	2	3	4	5	2
37.	A java servlet can write to the HTTP response header by calling the ___________ method of the HttpServlet Response object.	setStatus()	sendError()	sendRedirect()	None of the above	setStatus()
38.	The ___________ method is used to notify the client that an error has occurred.	setStatus()	sendError()	sendRedirect()	None of the above	sendError()
39.	The ____________ method transmits a location header to the browser.	setStatus()	sendError()	sendRedirect()	None of the above	sendRedirect()
40.	The HTTP Response Header ____________ is a parameter for the connection header.	close	Content-Encoding	Content-Language	Content-Length	close
41.	The HTTP Response Header ____________ indicates page encoding .	close	Content-Encoding	Content-Language	Content-Length	Content-Encoding

42.	The HTTP Response Header ___________ indicates the language of the document.	close	Content-Encoding	Content-Language	Content-Length	Content-Language
43.	The HTTP Response Header _____________ indicates the number of bytes in the message before any character encoding is applied.	close	Content-Encoding	Content-Language	Content-Length	Content-Length
44.	The HTTP Response Header ____________ indicates the MIME type of the response document.	Content-Type	Expires	Last-Modified	Location	Content-Type
45.	The HTTP Response Header ____________ specifies the time in milliseconds when document is out of date.use	Content-Type	Expires	Last-Modified	Location	Expires
46.	The HTTP Response Header ____________ indicates the last time the document was changed.	Content-Type	Expires	Last-Modified	Location	Last-Modified
47.	The HTTP Response Header ____________ indicates the location of the document.	Content-Type	Expires	Last-Modified	Location	Location
48.	183. The HTTP Response Header ____________ indicates the number of seconds to wait before asking for a page update.	Refresh	Retry-After	Set-Cookie	WWW-Authenticate	Refresh
49.	The HTTP Response Header ____________ indicates the number of seconds to wait before requesting service, if the service is unavailable.	Refresh	Retry-After	Set-Cookie	WWW-Authenticate	Retry-After

50.	The HTTP Response Header ___________ identifies the cookie for the page.	Refresh	Retry-After	Set-Cookie	WWW-Authenticate	Set-Cookie
51.	The HTTP Response Header ___________ indicates the authorization type.	Refresh	Retry-After	Set-Cookie	WWW-Authenticate	Retry-After
52.	A cookie is composed of ___________ pieces.	2	3	4	5	2
53.	The ___________ is used to identify a particular cookie from among other cookies stored at the client.	cookie name	cookie value	cookie API	Cookie parameter	cookie name
54.	The ___________ is associated with the cookies.	cookie name	cookie value	cookie API	Cookie parameter	cookie value
55.	A java servlet writes a cookie by passing the construction of the cookie object __________ arguments.	2	3	4	5	2
56.	The __________ method returns an array of cookie objects.	getCookie()	addCookie()	setValue()	SetCookie()	getCookie()
57.	A java servlet can modify the value of an existing cookies by using the ___________ method of the cookie object.	getCookie()	addCookie()	setValue()	SetCookie()	setValue()
58.	The _______ object contains methods that are used to copy data from the ResultSet into a java collection object or variable for further processing.	statement	preparedstatement	callableStatement	ResultSet	ResultSet

59.	The _________ parameter is a single parameter that is used to both pass information to the stored procedure and retrieve information from a stored procedure.	IN	OUT	INOUT	None of the above	INOUT
60.	The _________ parameter contains a value returned by the stored procedures. The future generation of computers?	IN	OUT	INOUT	None of the above	OUT
61.	The ________ mthod returns a Boolean true if the row contains data.	getString()	next()	execute()	d) close()	next()
62.	There are _______ methods of the ResultSet object that are used to position the virtual cursor.	3	4	5	6	6
63.	The _________ method moves the virtual cursor specified to the first row in the ResultSet.	first()	next()	relative()	absolute()	first()
64.	The _________ method moves the virtual cursor the specified number of rows contained in the parameter.	first()	next()	relative()	absolute()	relative()
65.	A value in a column of the ResultSet can be replaced with a NULL value by using the _________ method.	updataeNull()	updateRow()	updateString()	deleteRow()	updataeNull()

66.	The ________ method is used to change the value of the column of the ResultSet.	updataeNull()	updateRow()	updateString()	deleteRow()	updateString()
67.	The _________ method is used to remove a row from a ResultSet.	updataeNull()	updateRow()	updateString()	deleteRow()	deleteRow()
68.	The _______ method returns the product name of the database.	getDatabaseProduct Name()	getUserName( )	getURL()	getschemas( )	getDatabaseProd uctName()
69.	The ________ method returns the username.	getDatabaseProduct Name()	getUserName( )	getURL()	getschemas( )	getUserName()
70.	The _________ method returns the URL of the database.	getDatabaseProduct Name()	getUserName( )	getURL()	getschemas( )	getURL()
71.	The _________ method returns all the schema names available in the database.	getDatabaseProduct Name()	getUserName( )	getURL()	getschemas( )	getschemas()
72.	The __________ method returns primary keys.	getPrimaryKeys()	getProcedures ()	getTables()	getColumnC ount()	getPrimaryKeys()
73.	The ________ method returns stored procedure names.	getPrimaryKeys()	getProcedures ()	getTables()	getColumnC ount()	getProcedures()

74.	The _________ method returns names of table in the database.	getPrimaryKeys()	getProcedures()	getTables()	getColumnCount()	getTables()
75.	The __________ method returns the number of columns contained in the ResultSet.	getPrimaryKeys()	getProcedures()	getTables()	getColumnCount()	getColumnCount()

UNIT 4

ENTERPRISE JAVABEAN

OVERVIEW OF EJB

Enterprise beans are Java EE components that implement Enterprise JavaBeans (EJB) technology. Enterprise beans run in the EJB container, a runtime environment within the Application Server. Although transparent to the application developer, the EJB container provides system-level services such as transactions and security to its enterprise beans. These services enable you to quickly build and deploy enterprise beans, which form the core of transactional Java EE applications. Written in the Java programming language, an **enterprise bean** is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called `checkInventoryLevel` and `orderProduct`. By invoking these methods, clients can access the inventory services provided by the application.

BENEFITS OF ENTERPRISE BEANS

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services such as transaction management and security authorization.
Second, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.
Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant Java EE server provided that they use the standard APIs.

EJB DEPLOYMENT DESCRIPTOR

Deployment descriptor is the file which tells the EJB server that which classes make up the bean implementation, the home interface and the remote interface. it also indicates the behavior of one EJB with other. The deployment descriptor is generally called as ejb-jar.xml and is in the directory META-INF of the client application. In the example given below our application consists of single EJB Here the node

```
<?xml version ="1.0" encoding="UTF-8"?>
<application-client version="5" xmlns="http://java
.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application-client_5.xsd">
<description>Accessing Database Application</description>
```

```
<display-name>Secure-app-client</display-name><enterprise-beans>
<session>
<ejb-name>secure</ejb-name>
<home>org.glassfish.docs.secure.secureHome</home>
<remote>org.glassfish.docs.secure.secure</remote>
<ejb-class>org.glassfish.docs.secure.secureBean</ejb-class>
<session-type>Stateless</session-type>
</session>
</enterprise-beans>
</application-client>
```

<ejb-name>secure</ejb-name>:-This is the node that assigns the name to the EJB.
<description>Accessing Database Application</description>:-This node gives the brief description about the Ejb module created.
<session-type>Stateless</session-type>:-This node assigns the Session bean as stateless or stateful. Here stateless means to say accessing Remote interface.

DEPLOYING EJB TECHNOLOGY

The container handles persistence, transactions, concurrency, and access control automatically for the enterprise beans. The EJB specification describes a declarative mechanism for how these things will be handled, through the use of an XML deployment descriptor. When a bean is deployed into a container, the container reads the deployment descriptor to find out how transaction, persistence (entity beans), and access control should be handled. The person deploying the bean will use this information and specify additional information to hook the bean up to these facilities at run time. A deployment descriptor has a predefined format that all EJB-compliant beans must use and all EJB-compliant servers must know how to read. This format is specified in an XML Document Type Definition, or DTD. The deployment descriptor describes the type of bean (session or entity) and the classes used for the remote, home, and bean class. It also specifies the transactional attributes of every method in the bean, which security roles can access each method (access control), and whether persistence in the entity beans is handled automatically or is performed by the bean. Below is an example of a XML deployment descriptor used to describe the Customer bean:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
<enterprise-beans>
<entity>
<description>
This bean represents a customer
</description>
<ejb-name>CustomerBean</ejb-name>
<home>CustomerHome</home>
<remote>Customer</remote>
<ejb-class>CustomerBean</ejb-class>
<persistence-type>Container</persistence-type>
<prim-key-class>Integer</prim-key-class>
```

```
<reentrant>False</reentrant>
<cmp-field><field-name>myAddress</field-name></cmp-field>
<cmp-field><field-name>myName</field-name></cmp-field>
<cmp-field><field-name>myCreditCard</field-name></cmp-field>
</entity>
</enterprise-beans>
<assembly-descriptor>
   <security-role>
   <description>
   This role represents everyone who is allowed full access to the Customer bean.
   </description>
   <role-name>everyone</role-name>
   </security-role>
   <method-permission>
   <role-name>everyone</role-name>
   <method>
   <ejb-name>CustomerBean</ejb-name>
   <method-name>*</method-name>
   </method>
   </method-permission>
   <container-transaction>
   <description>
   All methods require a transaction
   </description>
   <method>
   <ejb-name>CustomerBean</ejb-name>
   <method-name>*</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
   </container-transaction>
   </assembly-descriptor>
   </ejb-jar>
```

SESSION BEAN

A **session bean** represents a single client inside the Application Server. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server.

As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. Like an interactive session, a session bean is not persistent. (That is, its data is not saved to a database.) When the client terminates, its session bean appears to terminate and is no longer associated with the client.

STATE MANAGEMENT MODES

There are two types of session beans: stateful and stateless.

Stateful Session Beans

The state of an object consists of the values of its instance variables. In a stateful session bean, the instance variables represent the state of a unique client-bean session. Because the client interacts (―talks‖) with its bean, this state is often called the conversational state. The state is retained for the duration of the client-bean session. If the client removes the bean or terminates, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends there is no need to retain the state.

As an example, the HotelClerk bean can be modified to be a stateful bean which can maintain conversational state between method invocations. This would be useful, for example, if you want the HotelClerk bean to be able to take many reservations, but then process them together under one credit card. This occurs frequently, when families need to reserve two or more rooms or when corporations reserve a block of rooms for some event.

Below the HotelClerkBean is modified to be a stateful bean:

```
import javax.ejb.SessionBean;
import javax.naming.InitialContext;
public class HotelClerkBean implements SessionBean {
InitialContext jndiContext;
//conversational-state
Customer cust;
Vector resVector = new Vector();
public void ejbCreate(Customer customer) {}
cust = customer;
}
public void addReservation(Name name, RoomInfo ri,
Date from, Date to) {
ReservationInfo resInfo =
new ReservationInfo(name,ri,from,to);
resVector.addElement(resInfo);
}
public void reserveRooms() {
CreditCard card = cust.getCreditCard();
Enumeration resEnum = resVector.elements();
while (resEnum.hasMoreElements()) {
ReservationInfo resInfo =
(ReservationInfo) resEnum.nextElement();
RoomHome roomHome = (RoomHome)
getHome("java:comp/env/ejb/RoomEJB", RoomHome.class);
Room room =
roomHome.findByPrimaryKey(resInfo.roomInfo.getID());
double amount = room.getPrice(resInfo.from,restInfo.to);
CreditServiceHome creditHome = (CreditServiceHome)
getHome("java:comp/env/ejb/CreditServiceEJB",
CreditServiceHome.class);
CreditService creditAgent = creditHome.create();
creditAgent.verify(card, amount);
ReservationHome resHome = (ReservationHome)
getHome("java:comp/env/ejb/ReservationEJB",
ReservationHome.class);
Reservation reservation =
resHome.create(resInfo.getName(),
resInfo.roomInfo,resInfo.from,resInfo.to);
}
public RoomInfo[] availableRooms(Location loc,
Date from, Date to) {
// Make an SQL call to find available rooms
}
private Object getHome(String path, Class type) {
Object ref = jndiContext.lookup(path);
return PortableRemoteObject.narrow(ref,type);
}}
```

In the stateful version of the HotelClerkBean class, the conversational state is the Customer reference, which is obtained when the bean is created, and the Vector of ReservationInfo objects.

By maintaining the conversational state in the bean, the client is absolved of the responsibility of keeping track of this session state. The bean keeps track of the reservations and processes them in a batch when the serverRooms() method is invoked.

To conserve resources, stateful session beans may be passivated when they are not in use by the client. Passivation in stateful session beans is different than for entity beans. In stateful beans, passivation means the bean conversational-state is written to a secondary storage (often disk) and the instance is evicted from memory. The client's reference to the bean is not affected by passivation; it remains alive and usable while the bean is passivated.

When the client invokes a method on a bean that is passivated, the container will activate the bean by instantiating a new instance and populating its conversational state with the state written to secondary storage. This passivation/activation process is often accomplished using simple Java serialization but it can be implemented in other proprietary ways as long as the mechanism behaves the same as normal serialization. (One exception to this is that transient fields do not need to be set to their default initial values when a bean is activated.) Stateful session beans, unlike stateless beans, do use the ejbActivate() and ejbPassivate() methods. The container will invoke these methods to notify the bean when it's about to be passivated (ejbPassivate()) and immediately following activation ejbActivate()). Bean developers should use these methods to close open resources and to do other clean-up before the instance's state is written to secondary storage and evicted from memory.

The ejbRemove() method is invoked on the stateful instance when the client invokes the remove() method on the home or remote interface. The bean should use the ejbRemove() method to do the same kind of clean-up performed in the ejbPassivate() method.

Stateless Session Beans

A stateless session bean does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client, but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. That is, the state of a stateless session bean should apply accross all clients.Because stateless session beans can support multiple clients, they can offer better scalability for applications that require large numbers of clients. Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients. A stateless session bean can implement a web service, but other types of enterprise beans cannot.

An example of a stateless session bean is a CreditService bean, representing a credit service that can validate and process credit card charges. A hotel chain might develop a CreditService bean to encapsulate the process of verifying a credit card number, making a charge, and recording the charge in the database for accounting purposes. Below are the remote and home interfaces for the CreditService bean:

```
// remote interface
public interface CreditService extends javax.ejb.EJBObject {
public void verify(CreditCard card, double amount)
throws RemoteException, CreditServiceException;
public void charge(CreditCard card, double amount)
throws RemoteException, CreditServiceException;
}
// home interface
public interface CreditServiceHome extends java.ejb.EJBHome {
public CreditService create()
throws RemoteException, CreateException;
}
```

The remote interface, CreditService, defines two methods, verify() and charge(), which are used by the hotel to verify and charge credit cards. The hotel might use the verify() method to make a reservation, but not charge the customer. The charge() method would be used to charge a customer for a room. The home interface, CreditServiceHome provides one create() method with no arguments. All home interfaces for stateless session beans will define just one method, a no-argument create() method, because session beans do not have find methods and they cannot be initiated with any arguments when they are created. Stateless session beans do not have find methods, because stateless beans are all equivalent and are not persistent. In other words, there is no unique stateless session beans that can be located in the database. Because stateless session beans are not persisted, they are transient services. Every client that uses the same type of session bean gets the same service.

Below is the bean class definition for the CreditService bean. This bean encapsulates access to the Acme Credit Card processing services. Specifically, this bean accesses the Acme secure Web server and posts requests to validate or charge the customer's credit card.

```
import javax.ejb.SessionBean;
public class CreditServiceBean implements SessionBean {
URL acmeURL;
HttpURLConnection acmeCon;
public void ejbCreate() {
```

```
try {
InitialContext jndiContext = new InitialContext();
URL acmeURL = (URL)
jndiContext.lookup("java:comp/ejb/env/url/acme");
acmeCon = acmeURL.openConnection();
}
catch (Exception e) {
throws new EJBException(e);
} }
public void verify(CreditCard card, double amount) {
String response = post("verify:" + card.postString() +
":" + amount);
if (response.substring("approved")== -1)
throw new CreditServiceException("denied");
}
public void charge(CreditCard card, double amount)
throws CreditCardException {
String response = post("charge:" + card.postString() +
":" + amount);
if (response.substring("approved")== -1)
throw new CreditServiceException("denied");
}
private String post(String request) {
try {
acmeCon.connect();
acmeCon.setRequestMethod("POST "+request);
String response = acmeCon.getResponseMessage();
}
catch (IOException ioe) {
throw new EJBException(ioe);
}
}
public void ejbRemove() {
acmeCon.disconnect();
}
public void setSessionContext(SessionContext cntx) {}
public void ejbActivate() {}
public void ejbPassivate() {}
}
```

WHEN TO USE SESSION BEANS

In general, you should use a session bean if the following circumstances hold:

- At any given time, only one client has access to the bean instance.
- The state of the bean is not persistent, existing only for a short period (perhaps a few hours).
- The bean implements a web service.

Stateful session beans are appropriate if any of the following conditions are true:

- The bean's state represents the interaction between the bean and a specific client.

- The bean needs to hold information about the client across method invocations.

The bean mediates between the client and the other components of the application, presenting a simplified view to the client.

To improve performance, choose a stateless session bean if it has any of these traits:

- The bean's state has no data for a specific client.
- In a single method invocation, the bean performs a generic task for all clients. For example, use a stateless session bean to send an email that confirms an online order

ENTITY BEANS

The entity bean is one of three primary bean types: entity, session and Message Driven. The entity Bean is used to represent data in the database. It provides an object-oriented interface to data that would normally be accessed by the JDBC or some other back-end API. More than that, entity beans provide a component model that allows bean developers to focus their attention on the business logic of the bean, while the container takes care of managing persistence, transactions, and access control.

There are two basic kinds of entity beans: container-managed persistence (CMP) and bean-managed persistence (BMP). With CMP, the container manages the persistence of the entity bean. With BMP, the entity bean contains database access code (usually JDBC) and is responsible for reading and writing its own state to the database.

CONTAINER-MANAGED PERSISTENCE

Container-managed persistence beans are the simplest for the bean developer to create and the most difficult for the EJB server to support. This is because all the logic for synchronizing the bean's state with the database is handled automatically by the container. This means that the bean developer doesn't need to write any data access logic, while the EJB server is supposed to take care of all the persistence needs automatically -- a tall order for any vendor. Most EJB vendors support automatic persistence to a relational database, but the level of support varies. Some provide very sophisticated object-to-relational mapping, while others are very limited.In this panel, you will expand the CustomerBean developed earlier to a complete definition of a Container-managed persistence bean. In the panel on bean-managed persistence, you will modify the CustomerBean to manage its own persistence.

BEAN CLASS

An enterprise bean is a complete component that is made up of at least two interfaces and a bean implementation class. All these types will be presented and their meaning and application explained, starting with the bean class, which is defined below:

```
        import javax.ejb.EntityBean;
public class CustomerBean implements EntityBean {
int customerID;
Address myAddress;
Name myName;
CreditCard myCreditCard;
// CREATION METHODS
public Integer ejbCreate(Integer id) {
```

```
customerID = id.intValue();
return null;
}
public void ejbPostCreate(Integer id) {
}
public Customer ejbCreate(Integer id, Name name) {
myName = name;
return ejbCreate(id);
}
public void ejbPostCreate(Integer id, Name name) {
}
// BUSINESS METHODS
public Name getName() {
return myName;
}
public void setName(Name name) {
myName = name;
}
public Address getAddress() {
return myAddress;
}
public void setAddress(Address address) {
myAddress = address;
}
public CreditCard getCreditCard() {
return myCreditCard;
}
public void setCreditCard(CreditCard card) {
myCreditCard = card;
}
// CALLBACK METHODS
public void setEntityContext(EntityContext cntx) {
}
public void unsetEntityContext() {
}
public void ejbLoad() {
}
public void ejbStore() {
}
public void ejbActivate() {
}
public void ejbPassivate() {
}
public void ejbRemove() {
}
}
```

Notice that there is no database access logic in the bean. This is because the EJB vendor provides tools for mapping the fields in the CustomerBean to the database. The

CustomerBean class, for example, could be mapped to any database providing it contains data that is similar to the fields in the bean. In this case, the bean's instance fields are composed of a primitive int and simple dependent objects (Name, Address,and CreditCard) with their own attributes Below are the definitions for these dependent objects:

```
// The Name class
public class Name implements Serializable {
public String lastName, firstName, middleName;
public Name(String lastName, String firstName,
String middleName) {
this.lastName = lastName;
this.firstName = firstName;
this.middleName =
middleName;
}
public Name() {}
}
// The Address class
public class Address implements Serializable {
public String street, city, state, zip;
public Address(String street, String city,
String state, String zip) {
this.street = street;
this.city = city;
this.state = state;
this.zip = zip;
}
public Address() {}
}
// The CreditCard class
public class CreditCard implements Serializable {
public String number, type, name;
public Date expDate; public CreditCard(String number, String type, String name, Date
expDate) {
this.number = number;
this.type = type;
this.name = name;
this.expDate = expDate;
}
public CreditCard() {}
}
```

These fields are called container-managed fields because the container is responsible for synchronizing their state with the database; the container manages the fields. Container-managed fields can be any primitive data types or serializable type. This case uses both a primitive int (customerID) and serializable objects (Address, Name, CreditCard). To map the dependent objects to the database, a fairly sophisticated mapping tool would be needed. Not all fields in a bean are automatically container-managed fields; some may be just plain instance fields for the bean's transient use. A bean developer distinguishes container-managed fields from plain instance fields by indicating which fields are container-managed in the deployment descriptor. The container-managed fields must have corresponding types (columns in RDBMS) in the database either directly or through object-relational mapping. The CustomerBean might, for example, map to a CUSTOMER table in the database that has the following definition:

```
CREATE TABLE CUSTOMER
{
id INTEGER PRIMARY KEY,
last_name CHAR(30),
first_name CHAR(20),
middle_name CHAR(20),
street CHAR(50),
city CHAR(20),
state CHAR(2),
zip CHAR(9),
credit_number CHAR(20),
credit_date DATE,
credit_name CHAR(20),
credit_type CHAR(10)
}
```

With container-managed persistence, the vendor must have some kind of proprietary tool that can map the bean's container-managed fields to their corresponding columns in a specific table, CUSTOMER in this case.

Once the bean's fields are mapped to the database, and the Customer bean is deployed, the container will manage creating records, loading records, updating records, and deleting records in the CUSTOMER table in response to methods invoked on the Customer bean's remote and home interfaces.
A subset (one or more) of the container-managed fields will also be identified as the bean's primary key. The primary key is the index or pointer to a unique record(s) in the database that makes up the state of the bean. In the case of the CustomerBean, the id field is the primary key field and will be used to locate the bean's data in the database. Primitive single field primary keys are represented as their corresponding object wrappers. The primary key of the Customer bean for example is a primitive int in the bean class, but to a bean's clients it will manifest itself as the java.lang.Integer type. Primary keys that are made up of several fields, called compound primary keys, will be represented by a special class defined by the bean developer. Primary keys are similar in concept to primary keys in a relational database -- actually when a relational database is used for persistence, they are often the same thing.
).

MESSAGE-DRIVEN BEAN

A message-driven bean is an enterprise bean that allows Java EE applications to process messages asynchronously. It normally acts as a JMS message listener, which is similar to an event listener except that it receives JMS messages instead of events.
The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use

Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

What Makes Message-Driven Beans Different from Session Beans?

The most visible difference between message-driven beans and session beans is that clients do not access message-driven beans through interfaces. In several respects, a message-driven bean resembles a stateless session bean.A message-driven bean's instances retain no data or conversational state for a specific client. All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently. A single message-driven bean can process messages from multiple clients.The instance variables of the message-driven bean instance can contain some state across the handling of client messages (for example, a JMS API connection, an open database connection, or an object reference to an enterprise bean object).Client components do not locate message-driven beans and invoke methods directly on them. Instead, a client accesses a message-driven bean through, for example, JMS by sending messages to the message destination for which the message-driven bean class is the MessageListener. You assign a message-driven bean's destination during deployment by using Application Server resources.

Message-driven beans have the following characteristics:

- They execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.
- They do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.
- They are stateless.

When a message arrives, the container calls the message-driven bean's onMessage method to process the message. The onMessage method normally casts the message to one of the five JMS message types and handles it in accordance with the application's business logic. The onMessage method can call helper methods, or it can invoke a session bean to process the information in the message or to store it in a database.

A message can be delivered to a message-driven bean within a transaction context, so all operations within the onMessage method are part of a single transaction. If message processing is rolled back, the message will be redelivered

WHEN TO USE MESSAGE-DRIVEN BEANS

Session beans allow you to send JMS messages and to receive them synchronously, but not asynchronously. To avoid tying up server resources, do not to use blocking synchronous receives in a server-side component, and in general JMS messages hould not be sent or received synchronously. To receive messages asynchronously, use a message-driven bean.

Example For Message Driven Bean

Example Application Overview

This application has the following components:

- SimpleMessageClient: A J2EE application client that sends several messages to a queue.
- SimpleMessageEJB: A message-driven bean that asynchronously receives and processes the messages that are sent to the queue.

Figure 4.14.1 illustrates the structure of this application. The application client sends messages to the queue, which was created administratively using the j2eeadmin command. The JMS provider (in this, case the J2EE server) delivers the messages to the instances of the message-driven bean, which then processes the messages.

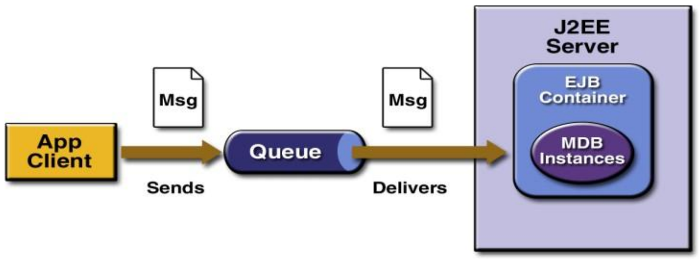


Figure 4.14.1 The SimpleMessageApp Application

KEY TERMS

- **Enterprise JavaBeans**: Enterprise bean implements a business task, or a business entity.
- **EJB Server and Container**: An EJB bean is said to reside within an EJB Container that in turn resides within an EJB Server.
- **Deployment descriptors**: The additional information required to install an EJB within its server is provided in the deployment descriptors for that bean
- **The EJBObject:** An instance of a generated class that implements the remote interface defined by the bean developer
- **The EJBLocalObject**: An instance of a generated class that implements the local interface defined by the bean developer

Question and Excersice

Short Answer Questions

1. Differentiate Java Bean and EJB
2. List the three different Types of EJB Classes
3. Define Callback Method
4. What is jar file?

Long –Answer Question

1. Explain the Entity bean with example
2. Describe in Detail about Session bean
3. Illustrate Message Drive bean with suitable example

Reference

1. **J**im Keogh , ―The Complete Reference J2SE‖, 1st Edition, Tata McGraw Hill Edition, New Delhi:2002.
2. http://publib.boulder.ibm.com/infocenter/wbihelp/v6rxmx/index.jsp? topic=/com.ibm.wics_developer.doc/doc/access_dev_ejb/access_dev_ejb16 .htmDeepak Vohra , JDBC 4.0 and Oracle JDeveloper for J2EE Development
3. Derek Ashmore , The J2EE Architect's Handbook
4. JSP - http://java.sun.com/products/jsp/docs.html

UNIT-IV
J2EE

S. No.	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	The EJB _________ is a vendor provided entity located on the EJB server that manages system-level services for EJB.	container	classes	interfaces	none of the above	container
2	There are ________ kinds of EJB types. Information	2	3	4	5	3
3	The session and entity beans must have ___________ interfaces.	2	3	4	5	2
4	A __________ is used to represent business data.	entity bean	session bean	message-driven bean	none of the above	entity bean
5	A _________ bean is used to model a business process.	entity	session	message-driven	none of the above	session
6	A _________ bean is used to receive messages from a JMS resource.	entity	session	message-driven	none of the above	message-driven
7	The ________ handles communication between the EJB and other components in the EJB environment using the Home interface and the Remote interface.	EJB container	EJB classes	EJB interfaces	deployment descriptors	EJB container

8	A ________ describes how EJBs are managed at runtime and enables the customization of EJB behavior without modification to the EJB code.	EJB container	EJB classes	EJB interfaces	deployment descriptors	deployment descriptors
9	A ____________ is written in a file using XML syntax.	EJB container	EJB classes	EJB interfaces	deployment descriptors	deployment descriptors
10	The expansion of IDE is ____________.	Integral Development Environment	Integrated Development Environment	Integrity Development Environment	Industrial development environment	Integrated Development Environment
11	The ____________ file is packages in the Java Archive file along with the other files that are required to deploy the EJB.	EJB container	EJB classes	EJB interfaces	deployment descriptors	deployment descriptors
12	The ________ element is the root element of the deployment descriptor.	<ejb-jar>	<ejb-name>	<ejb-class>	<entity>	<ejb-jar>
13	There are _________ elements that are contained within the <enterprise-beans> element.	2	3	4	5	3
14	The first element within the <ejb-jar> element is the _________ element.	<enterprise-beans>	<home>	<local>	<ejb-class>	<enterprise-beans>
15	The ____________ element contains subelements that describe the entity EJB.	<enterprise-beans>	<home>	<local>	<entity>	<entity>

16	The ________ element describes the fully qualified class name of the Remote interface, which defines the entity EJB's business mthods to remote clients.	<remote >	<local-home>	<reentrant>	<persistence-type>	<remote >
17	The _________ element defines how the entity EJB manages persistence.	<remote >	<local-home>	<reentrant>	<persistence-type>	<persistence-type>
18	The ________ element declares whether or not an entity EJB can be looped back without throwing an exception.	<remote>	<reentrant>	<ejb-class>	<remote>	<reentrant>
19	The subelement ___________ describes the deployment descriptor.	<description>	<display-name>	<small-icon>	<large-icon>	<description>
20	The subelement ________ describes the JAR file and individual EJB components.	<description>	<display-name>	<small-icon>	<large-icon>	<display-name>
21	The subelement _________ describes one or more enterprise beans contained in the JAR file.	<enterprise-beans>	<ejb-client-jar>	<assembly-descriptor>	<description>	<enterprise-beans>
22	The subelement ___________ describes the path of the client JAR and is used by the client to access EJBs described in the deployment descriptor.	<enterprise-beans>	<ejb-client-jar>	<assembly-descriptor>	<description>	<ejb-client-jar>
23	The subelement ___________ describes how EJBs are used in the J2EE application.	<enterprise-beans>	<ejb-client-jar>	<assembly-descriptor>	<description>	<assembly-descriptor>

24	The subelement ________ describes a small icon within the jar file that is used to represent the JAR file.	<description>	<small-icon>	<display-name>	<large-icon>	<small-icon>
25	There subelement _________ describes the fully qualified class name of the session or entity EJB remote interface.	<remote>	<local-home>	<local >	<ejb-class >	<remote>
26	The subelement _________ describes the primary key filed for entity beans that use container-managed persistence.	<primary-field>	<prim-key-class>	<persistence-type>	<local>	<primary-field>
27	The subelement ___________ specifies the version of container-managed persistence.	<reentrant >	<cmp-version>	<cmp-field>	<env-entry>	<cmp-version>
28	The ________ element is used to specify an EJB's security role.	<security-role-ref>	<role-name>	<role-link>	<description>	<security-role-ref>
29	A _________ is used in a deployment descriptor to specify a query method and a QL statement that is used as the criteria for selecting data from a relational database.	<query>	<method-param>	<ejb-ql>	<query-method>	<query-method>
30	The _________ subelement itself has two subelements.	<query>	<method-param>	<ejb-ql>	<query-method>	<query-method>
31	The ___________ subelement specifies the name of the method.	<query>	<method-param>	<ejb-ql>	<method-name>	<method-name>
32	The ________ subelement of the <query> element contains a SQL statement that is used to retrieve information from the database.	<ejb-ql>	<query>	<query-method>	<method-param>	<ejb-ql>

33	There are _________ types of cardinality relationships.	2	3	4	5	4
34	The cardinality relationships has one of _________ directions.	2	3	4	5	2
35	A ____________ is to execute a unit of work that may involve multiple tasks.	transaction	method	assembly	attribute	transaction
36	The ________ method is called whenever the session bean is removed from the pool and is referenced by a client.	ejbActivate()	ejbPassivate()	ejbRemove()	ejbCreate()	ejbActivate()
37	The _________ method is called before the instance enters the "passive" state when the session bean is returned to the object pool and should contain routines that release resources.	ejbActivate()	ejbPassivate()	ejbRemove()	ejbCreate()	ejbPassivate()
38	The _________ method is called just before the bean is available for garbage collection.	ejbActivate()	ejbPassivate()	ejbRemove()	ejbCreate()	ejbRemove()
39	The ____________ method is a method that contains business logic that is customized to the service provided by the EJB.	ejbActivate()	ejbPassivate()	ejbRemove()	myMethod()	myMethod()
40	A ________ is considered the powerhouse of a J2EE application.	entity java bean	session java bean	message-driven bean	none of the above	entity java bean
41	Data collected and managed by an entity bean is called ___________.	data	persistent data	information	none of the above	persistent data

42	There are __________ groups of methods that are typically contained in an entity bean.	2	3	4	5	3
43	There are ______ commonly used callback methods.	4	5	64	7	7
44	The ________ method is called immediately following the creation of the instance and sets the content that is associated with the entity.	setEntityContext ()	unsetEntityContext ()	ejbLoad()	ejbStore()	setEntityContext ()
45	The _________ method is called whenever the instance of the entity bean is activated from its “passive” state.	setEntityContext ()	unsetEntityContext ()	ejbLoad()	ejbActivate()	ejbActivate()
46	A container invokes the _________ method to instruct the instance to synchronize its state by loading its state from the underlying database.	setEntityContext ()	unsetEntityContext ()	ejbLoad()	ejbActivate()	ejbLoad()
47	The ____________ method is invoked by a container to instruct the instance to synchronize its state by storing it to the underlying database.	setEntityContext ()	unsetEntityContext ()	ejbLoad()	ejbStore()	ejbStore()
48	The ________ method is called before the instance enters the “passive” state and should contain routines that release resources.	ejbPassivate()	ejbActivate()	ejbRemove()	ejbLoad()	ejbPassivate()

49	Thee _________ method is called immediately before the entity terminates by either the client or by the EJB container.	ejbPassivate()	ejbActivate()	ejbRemove()	ejbLoad()	ejbRemove()
50	There are _________ methods defined in a BMP bean.	2	3	4	5	5
51	In BMP bean, __________ method must contain code that reads data from a database.	ejbLoad()	ejbstore()	ejbCreate()	ejbRemove()	ejbLoad()
52	In BMP bean, the __________ method must have code that inserts a new record in a database.	ejbLoad()	ejbstore()	ejbCreate()	ejbRemove()	ejbCreate()
53	In BMP bean, the __________ method writes data to a database.	ejbLoad()	ejbstore()	ejbCreate()	ejbRemove()	ejbstore()
54	The _________ method is where the MBD processes messages received indirectly from a client.	onMessage()	getText()	ejbRemove()	setMessageDrivenContext()	onMessage()

UNIT 5

JAVA SERVER PAGES & JAVA RMI

BENEFITS OF JSP

One of the main reasons why the Java Server Pages technology has evolved into what it is today and it is still evolving is the overwhelming technical need to simplify application design by separating dynamic content from static template display data. Another benefit of utilizing JSP is that it allows to more cleanly separating the roles of web application/HTML designer from a software developer. The JSP technology is blessed with a number of exciting benefits, which are chronicled as follows:

1. The JSP technology is platform independent, in its dynamic web pages, its web servers, and its underlying server components. That is, JSP pages perform perfectly without any hassle on any platform, run on any web server, and web-enabled application server. The JSP pages can be accessed from any web server.

2. The JSP technology emphasizes the use of reusable components. These components can be combined or manipulated towards developing more purposeful components and page design. This definitely reduces development time apart from the At development time, JSPs are very different from Servlets, however, they are precompiled into Servlets at run time and executed by a JSP engine which is installed on a Web-enabled application server such as BEA WebLogic and IBM WebSphere

JSP ARCHITECTURE

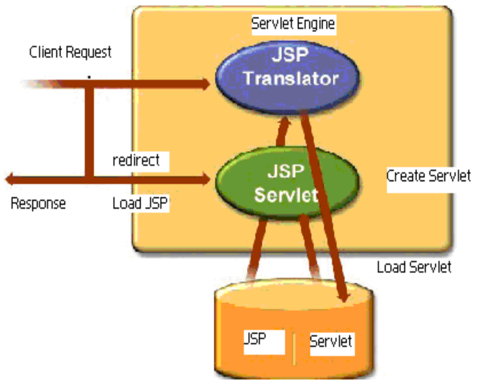
JSP pages are high level extension of servlet and it enables the developers to embed java code in html pages. JSP files are finally compiled into a servlet by the JSP engine. Compiled servlet is used by the engine to serve the requests.

javax.servlet.jsp package defines two interfaces:

- JSPPage
- HttpJspPage

These interfaces define the three methods for the compiled JSP page. These methods are:

- jspInt() – Called when JSP in requested
- jspDestroy() - Called when JSP is terminated
- jspService(HttpServletRequest request , HttpServletResponse response)-



JSP Architecture

The jspInt() method is identical to the init() method in a Java servlet and in an applet. The jspInt() method is called first when the JSP is requested and is used to initialize objects and variables that are used throughout the life of the JSP.

The jspDestroy() method is identical to the destroy method in a java servlet. The destroy() method is automatically called when the JSP terminates normally. The destroy() method is used for cleanup where resources used during the execution of the JSP are released, such as disconnecting from a database. The jspService() method is automatically called and retrieves a connection to HTTP.

JSP TAGS

A JSP program consists of a combination of HTML tags and JSP tags. JSP tags define java code that is to be executed before the output of the jsp program is sent to the browser.

A JSP tag begins with a <%, which is followed by Java code and ends with %>. There is also and Extendable Markup Language (XML) version of JSP tags, which are formatted as <jsp:TagID></JSP:TagID>.

In JSP tags can be divided into 5 different types. These are:

1. **Comment Tag:** A comment tag opens with <%-- and closes with --%>, and is followed by a comment that usually describes the functionality of statements that follow the comment tag.
2. **Directives tag:** In the directives we can import packages, define error handling pages or the session information of the JSP page.
3. **Declarations tag:**This tag is used for defining the functions and variables to be used in the JSP.
4. **Scriplets:** In this tag we can insert any amount of valid java code and these codes are placed in *_jspService* method by the JSP engine.
5. **Expressions:** An expression tag opens with <%= and is used for an expression statement whose result replaces the expression statement whose result replaces the expression tag when the JSP virtual engine resolves JSP tags. An expression tags close with %>

JSP Directives

Syntax of JSP directives is:

<%! //java codes %>

JSP Declaratives begins with <%! and ends %> with .We can embed any amount of java code in the JSP Declaratives. Variables and functions defined in the declaratives are class level and can be used anywhere in the JSP page

<%@directive attribute="value" %>

Where **directive** may be:

- page: page is used to provide the information about it.
  Example: <%@page language="java" %>
- include: include is used to include a file in the JSP page.
  Example:<%@ include file="/header.jsp" %>
- taglib: taglib is used to use the custom tags in the JSP pages (custom tags allows us to defined our own tags)
  Example: <%@ taglib uri="tlds/taglib.tld" prefix="mytag" %>

and **attribute** may be:

- language="java"
  This tells the server that the page is using the java language. Current JSP specification supports only java language.
  Example: <%@page language="java" %>
- extends="mypackage.myclass"
  This attribute is used when we want to extend any class. We can use comma(,) to import more than one packages.

Example:

<%@page language="java"import="java.sql.*,mypackage.myclass" %>

- session="true"
  When this value is true session data is available to the JSP page otherwise not. By default this value is true.
  Example: <%@page language="java" session="true" %>

- errorPage="error.jsp"
  errorPage is used to handle the un-handled exceptions in the page.
  Example: <%@page language="java" session="true" errorPage="error.jsp"%>

- contentType="text/html;charset=ISO-8859-1"
  Use this attribute to set the MIME type and character set of the JSP.
  Example:<%@page language="java" session="true" contentType="text/html; charset=ISO-8859-1" %>
- errorPage="error.jsp"
  errorPage is used to handle the un-handled exceptions in the page.
  Example: <%@page language="java" session="true" errorPage="error.jsp"%>

- contentType="text/html;charset=ISO-8859-1"
  Use this attribute to set the MIME type and character set of the JSP.
  Example:<%@page language="java" session="true" contentType="text/html; charset=ISO-8859-1" %>

Example:

```
<%@page contentType="text/html" %>
<html>
<body><%!
int cnt=0;
private int getCount(){
//increment cnt and return the value
cnt++;
return cnt;
}
%>
<p>Values of Cnt are:</p>
<p><%=getCount()%></p>

<p><%=getCount()%></p>
<p><%=getCount()%></p>
<p><%=getCount()%></p>
<p><%=getCount()%></p>
<p><%=getCount()%></p>
</body>
</html>
```

JSP SCRIPTLRTS

```
<% //java codes  %>
```

JSP Scriptlets begins with <% and ends %> .We can embed any amount of java code in the JSP Scriptlets. JSP Engine places these code in the _jspService() method. Variables available to the JSP Scriptlets are:

- **request:** request represents the clients request and is a subclass of HttpServletRequest. Use this variable to retrieve the data submitted along the request.
  Example:

```
<%//java codes
String userName=null; serName=request.getParameter("userName");    %>
```

- response: response is subclass of HttpServletResponse.
- session: session represents the HTTP session object associated with the request.
- out: out is an object of output stream and is used to send any output to the client.

Other variable available to the scriptlets are pageContext, application,config and exception.

JSP EXPRESSIONS

Syntax of JSP Expressions are:

```
<%="Any thing"  %>
```

JSP Expressions start with Syntax of JSP Scriptles are with <%= and ends with %>. Between these this you can put anything and that will convert to the String and that will be displayed.

Example:
<%="Hello World!" %>
Above code will display 'Hello World!'.

Display current time using Date class
- Current time: <%= new java.util.Date() %>

Display random number using Math class
- Random number: <%= Math.random() %>

Use implicit objects
- Your hostname: <%= request.getRemoteHost() %>
- Your parameter: <%= request.getParameter(–yourParameter‖) %>
- Server: <%= application.getServerInfo() %>

Session ID: <%= session.getId() %>

VARIABLES AND OBJECTS

In JSP variable can be declared same as in java. But the declaration statement must appear as a JSP tag within the JSP program before the variable or object used in the program.

Declaring and using a variable

```
<HTML>
  <HEAD>
    <TITLE>Creating a Variable</TITLE>
  </HEAD>
  <BODY>
    <H1>Creating a Variable</H1>
    <%
        int days= 365;%>
    <<p> Number of days = <%= days %></p>
  </BODY>
</HTML>
```

The variable days is used in an expression tag that is embedded within the HTML paragraph tag <p>. A JSP expression tag begins with <%=, which is followed by the expression. The JSP virtual engine resolves the JSP expression before sending the output of the JSP program to the browser. That is, the JSP tag <%=days%> is replaced with the number 365, afterwards, the HTML paragraph tag and related information is sent to the browser. It is able to place multiple statements with in a JSP tag by extending the close JSP program. This is illustrated in the below example where three variables are declared.

```
<HTML>
  <HEAD>
    <TITLE>Creating a Variables</TITLE>
  </HEAD>
  <BODY>
    <%
        int days= 365;
       int month=12;
           int weeks;
    %>
    <<p> Number of days = <%= days %></p>
  </BODY>
</HTML>
```

Array is used to store similar type of data in series. E.g. fruits name. Fruits can be a mango, banana, and apple. Name of students in classroom denote to 10th Standard, Bachelor in science can have group of 30 to 40 students. Arrays can be String array, int array, and dynamic arrays are ArrayList, vector.

The following program shows the JSP program create three String objects,

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<% String[] stArray={"bob","riche","jacky","rosy"};
%>
<html>
<body>
  <%
   int i=0;
   for(i=0;i<stArray.length;i++)
   {
    out.print("stArray Elements       :"+stArray[i]+"<br/>");
   }
  %>
</body>
</html>
```

This String Array has four elements. When we go through this array, have to use loop either for or while loop. We are using here for loop, First stArray.length give use total number of elements in array then we fetch one by one for loop iterator. Array starts from zero so here we have only 0,1,2,3 elements if we try to get stArray[4] it will throw

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<%
String[] stArray=new String[4];
stArray[0]="bob";
stArray[1]="riche";
stArray[2]="jacky";
stArray[3]="rosy";
%>
<html>
<body>
<%
   int i=0;
   for(i=0;i<stArray.length;i++)
   {
    out.print("stArray Elements       :"+stArray[i]+"<br/>");
   }
  %>
</body>
</html>
```

Integer Array in JSP

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>
<%
int[] intArray={23,45,13,54,78};
%>
<html>
<body>
   <%
    int i=0;
    for(i=0;i<intArray.length;i++)
    {
     out.print("intArray Elements       :"+intArray[i]+"<br/>");
    }
   %>
</body> </html>
```

Dynamic arrays are automatically growable and reduceable according to per requirement. We don't need to define it size when declaring array. It takes extra ratio of capacity inside memory and keeps 20% extra Vector ArrayList

vectorArray.jsp

```
<%@ page import="java.util.Vector" language="java" %>
<%
Vector vc=new Vector();
vc.add("bob");
vc.add("riche");
vc.add("jacky");
vc.add("rosy");
%>
<html>
<body>
   <%
    int i=0;
    for(i=0;i<vc.size();i++)
    {
     out.print("Vector Elements       :"+vc.get(i)+"<br/>");
    }
   %>
</body>
</html>
```

ArrayList: ArrayList also same just it is unsynchronized, unordered and faster than vector.

ArrayList.jsp

```
<%@ page import="java.util.ArrayList" language="java" %>
<%
ArrayList ar=new ArrayList();
ar.add("bob");
ar.add("riche");
ar.add("jacky");
ar.add("rosy");
```

```
%>
<html>
<body>
   <%     int i=0;
    for(i=0;i<ar.size();i++)
    {      out.print("ArrayList Elements       :"+ar.get(i)+"<br/>");      }
   %>
</body> </html>
```

JSP METHODS

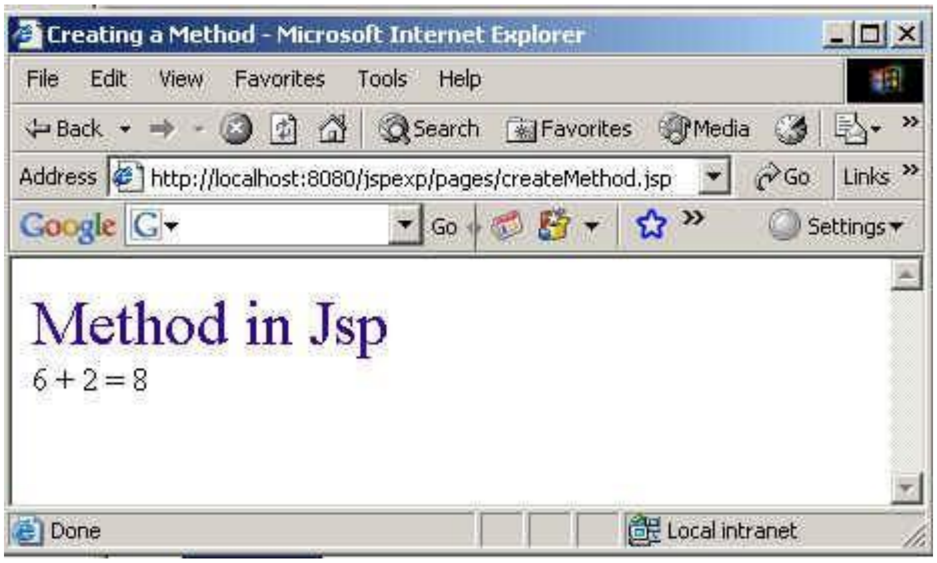
JSP offers the same versatility that have with JSP programs, such as defining methods that are local to the JSP program. A method is defined similar to how a method is defined in a java program except the method definition is place with in a JSP tag.Once the method is defined it can be called within the JSP tag.

In this below example it shows how to declare a method and how to use it. In this example making a method named as **addNum(int i, int b)** which will take two numbers as its parameters and return integer value. The method is declared inside a declaration directive i.e. <%! ----------- %> this is a declaration tag. This tag is used mainly for declaration the variables and methods. In the method adding to numbers is performed. To print the content of the method we are using scriptlet tag inside which we are going to use the out implicit object. <% ------- %> This tag is known as Scriptlets. The main purpose of using this tag is to embed a java code in the jsp page.

The code of the program is given below:

```
<% < HTML >
  <HEAD>
   <TITLE>Creating a Method</TITLE>
  </HEAD>
  <BODY>
<font size="6" color ="#330099"> Method in Jsp </ font ><br>
   <%!
   int addNum(int n, int m)
   {
     return n + m;
   }
   %>
```

Output of the program is given below:


Creating a Method - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/jspexp/pages/createMethod.jsp
Go Links
Google Go Settings
Done
Local intranet

Method in Jsp

6 + 2 = 8

A JSP program is capable of handling practically any kind of method that normally use in a Java program. The following example shows how to define and is an overloaded method.

Both methods are defined in the same JSP tags, although each follows Java Syntax structure for defining a method. One method uses a default value for the curve, while the overloaded method enables the statement that calls the method to provide the value of the curve.

Once again, these methods are called form an embedded JSP tag placed inside two HTML paragraph tags.

```
<HTML>
<HEAD>
<TITLE> JSP Programming</TITLE>
</HEAD>
<BODY>
<%! boolean curve (int grade)
        {
                return 10 + grade;
        }
boolean curve (int grade, int curveValue)
        {
                return curveValue + grade;
        }
%>
<p> your curve grade is : <%=curve(80,100)%></p>
<p> your curve grade is : <%=curve(70)%></p>
</BODY>

  </HTML>
```

CONTROL STATEMENTS

One of the most powerful features available in JSP is the ability to change the flow of the program to truly create dynamic content for a web page based on conditions received form the browsers.

If Statement

There are two control statements used to change the flow of a JSP program. These are the if statement and the switch statement, both of which are also used to direct the flow of a java program. The if statement evaluates a condition statement to determine if one or more lines of code are to be executed or skipped.

The if statement requires three JSP tags. The first contains the beginning of the if statement, including the conditional expression. The second contains the else statement, and the third has the closed French brace used to terminate the else block.

Example of if-else condition

ifelse.jsp

```
 <%@ page language="java" import="java.sql.*" %>
<html>
<head>
<title>while loop in JSP</title>
</head>
<body>
<%
String sName="joe";
String sSecondName="noe";
   if(sName.equals("joe")){
     out.print("if condition check satisfied JSP count :"+sName+"<br>");
   }

     if(sName.equals("joe") && sSecondName.equals("joe"))

{
     out.print("if condition check if Block <br>");
   }
   else
   {
     out.print("if condition check else Block <br>");
   }
%>
</body>
</html>
```

Using an if-else Ladder

```
 <HTML>
   <HEAD>
   <TITLE>Using an if-else Ladder</TITLE>
  </HEAD>
  <BODY>
   <H1>Using an if-else Ladder</H1>
   <%
      String day = "Friday";
      if(day == "Monday")
         out.println("It\'s Monday.");
      else if (day == "Tuesday")
         out.println("It\'s Tuesday.");
      else if (day == "Wednesday")
         out.println("It\'s Wednesday.");
      else if (day == "Thurssday")
         out.println("It\'s Thursday.");
      else if (day == "Friday")
         out.println("It\'s Friday.");
      else if (day == "Saturday")
         out.println("It\'s Saturday.");
      else if (day == "Sunday")
         out.println("It\'s Sunday.");
   %>
  </BODY>
</HTML>
```

Switch Statement

A switch statement compares a value with one or more other values associated with a case statement. The code segment that is associated wit the matching case statement is executed. Code segments associated with other case statements are ignored.

```
<HTML>
  <HEAD>
    <TITLE>Using the switch Statement</TITLE>
  </HEAD>
  <BODY>
   <H1>Using the switch Statement</H1>
   <%
      int day = 3;

      switch(day) {
         case 0:
            out.println("It\'s Sunday.");
            break;
         case 1:
            out.println("It\'s Monday.");
            break;
         case 2:
            out.println("It\'s Tuesday.");
            break;
            case 3:
                out.println("It\'s Wednesday.");
                break;
            case 4:
                out.println("It\'s Thursday.");
                break;
            case 5:
                out.println("It\'s Friday.");
                break;
            default:
                out.println("It must be Saturday.");
        }
    %>
  </BODY>
</HTML>
```

LOOPS

There are three kinds of loops commonly used in a JSP program. These are the for loop, while loop, and the do…while loop.

For Loop:
The for loop repeats usually a specified number of times

Example of for loop in JSP

for.jsp

```
<%@ page language="java" import="java.sql.*" %>
<html>
<head>
<title>For loop in JSP</title>
</head>

<body>
<%
for(int i=0;i<=10;i++)
{
  out.print("Loop through JSP count :"+i+"<br/>");
}
%>
</body>
</html>
```

While Loop: The while loop executes continually as long as a specified condition remains true. However, the while loop may not execute because the condition may never be true. In contrast the do…while loop executes at least once; then, the conditional expression in the do… while loop is evaluated to determine if the loop should be executed another time.

Example of while loop in JSP

while.jsp

```
  <%@ page language="java" import="java.sql.*" %>
<html>
<head>
<title>while loop in JSP</title>
</head>
```

```
<body>
<%
int i=0;
while(i<=10)
{
  out.print("While Loop through JSP count :"+i+"<br/>");
  i++;
}
%>
</body>
</html>
```

Example of do-while loop in JSP

doWhile.jsp

```
  <%@ page language="java" import="java.sql.*" %>
<html>
<head>
<title>do-while loop in JSP</title>
</head>

<body>
<%
int i=0;
do{
  out.print("While Loop through JSP count :"+i+"<br/>");
  i++;
}
while(i<=10);
%>
</body></html>
```

RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

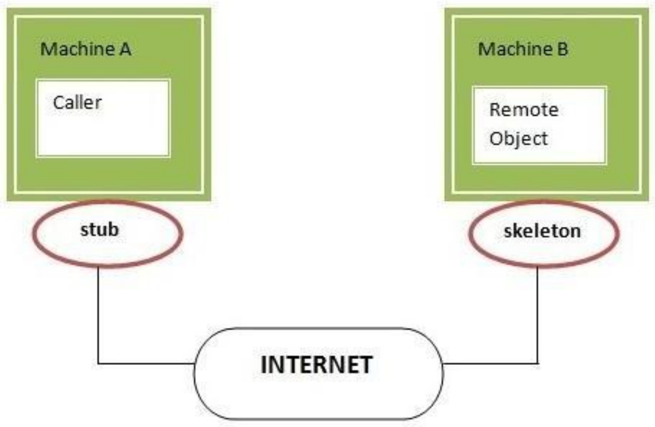
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

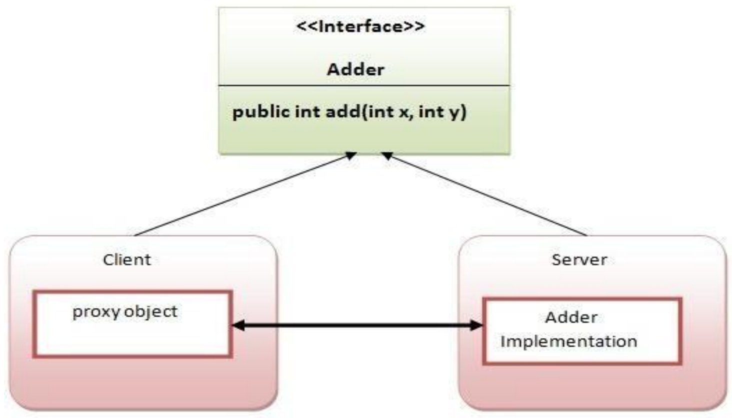
Java RMI Example

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

---

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

---

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

For running this rmi example,

1) compile all the java files

```
javac *.java
```

2)create stub and skeleton object by rmic tool

```
rmic AdderRemote
```

3)start rmi registry in one command prompt

```
rmiregistry 5000
```

4)start the server in another command prompt

```
java MyServer
```

5)start the client application in another command prompt java MyClient

KEY TERMS

- **Java Server Pages (JSP)**: JSP is a java based technology used for delivering dynamic content to web clients in a portable, secure and well-defined way
- **JSP tags:** define java code that is to be executed before the output of the JSP program is sent to the browser.
- **Comment Tag:** It is a tag opens with <%-- and closes with -- %>
- **Directives tag:** In the directives we can import packages, define error handling pages or the session information of the JSP page.
- **Declarations tag:** This tag is used for defining the functions and variables to be used in the JSP.

QUESTIONS AND EXERCISES

Short Answer Questions

1. List the benefits of JSP
2. Define the various methods used in JSP
3. What is JSP Directives and Expressions
4. What are the processes that participate in supporting remote method invocation?
5. Give a short notes on Variables and Objects in JSP
6. Define Tomcat

Long –Answer Question

1. Describe about JSP tags with suitable example
2. Illustrate Control structures in JSP with suitable example
3. How to Download and install Tomcat
4. Define Cookies? Write a JSP program to store and retrieve cookie information
5. Explain in detail about the Remote Method Invocation

References

5. Jim Keogh , ―The Complete Reference J2SE‖, 1st Edition, Tata McGraw Hill Edition, New Delhi:2002.
6. Marty Hall and Larry Brown , Core Servlets and JSP , 2nd Edition,Java 2 Platform, Enterprise Edition Series
7. Bruce Perry , Java Servlet & JSP Cookbook, O'Reilly
8. http://www.roseindia.net/jsp/jsparchitecture.shtml
9. www.pdf-search-engine.com/jsp-tutorial-pdf.html
10. www.jsptut.com/
11. www.roseindia.net/jsp/jsp-pdf-books.shtml

UNIT-IV
J2EE

S. No.	Question	Option 1	Option 2	Option 3	Option 4	Answer
1	A ____________ is called by a client to provide a web service, the nature of which depends on the J2EE application.	servlet	JSP classes	EJB	EIS	JSP classes
2	There are __________ methods that are automatically called when a JSP is requested and when the JSP terminates normally.	2	3	4	5	3
3	The _________ method is called first when the JSP is requested and is used to initialize objects and variables that are used throughout the life of the JSP.	jspInt()	jspDestroy	service()	jspProcess()	jspInt()
4	The _________ method is automatically called when the JSP terminates normally.	jspInt()	jspDestroy	service()	jspProcess()	jspDestroy
5	The ____________ method is automatically called and retrieves a connection to HTTP.	jspInt()	jspDestroy	service()	none of the above	service()
6	There are __________ factors that we must address when installing a JSP.	2	3	4	5	3
7	_________ tags define java code that is to be executed before the output of the JSP program is sent to the browser.	JSP	HTML	XML	None of the above	JSP

8	A JSP tag begins with a _________.	</	<*	<%	<!	<%
9	A JSP tag ends with a __________.	/>	*>	%>	!>	%>
10	There are __________ types of JSP tags.	2	3	4	5	5
11	A ___________ tag opens with <%-- and closes with --%>.	comment	declaration statement	directive	expression	comment
12	A _________ tag opens with <%!.	comment	declaration statement	directive	expression	declaration statement
13	A _________ tag opens with <%@.	comment	declaration statement	directive	expression	directive
14	A _________ tag opens with <%=.	comment	declaration statement	directive	expression	expression
15	A _________ tag opens with <%.	comment	declaration statement	directive	scriptlet	scriptlet
16	There are _________ kinds of loops commonly used in a JSP program.	2	3	4	5	3
17	The ____________ loop repeats usually a specified number of times.	for	while	do...while	none of the above	for
18	The ________ loop executes continuously as long as a specified condition remains true.	for	while	do...while	none of the above	while
19	The _________ loop executes atleast once.	for	while	do...while	none of the above	do...while
20	The _________ is the method used to parse a value of a specific field.	getParameter()	getParameterValues()	jspInit()	jspService()	getParameter()
21	There are ____________ predefined implicit objects that are in every JSP program.	2	3	4	5	4

22	There are ________ commonly used methods to track a session.	2	3	4	5	3
23	A JSP database system is able to share information among JSP programs within a _________ by using a session object.	servlet	session	EJB	none of the above	session
24	There are ________ steps necessary to make an object available to remote clients.	2	3	4	5	3
25	Method invoked by the client is called ___________.	server method	client method	RMI method	None of the above	client method
26	In addition to the methods that can be invoked by remote clients, the developer must also define other methods that support the processing of client-invoked methods. They are referred as _________.	server method	client method	RMI method	None of the above	server method
27	In RMI, port number _________ is the default port.	1099	1199	1299	1399	1099
28	The ________ method is used to locate the remote object.	myMethod()	lookup()	catch()	getMessage()	lookup()
29	The __________ method returns a String object that is passed to the println() method.	myMethod()	lookup()	catch()	getMessage()	myMethod()
30	Any exceptions that are thrown while the client-side program runs are trapped by the _________ block.	myMethod()	lookup()	catch()	getMessage()	catch()

31	The _________ calls the getMessage() method to retrieve the error message that is associated with the exception	myMethod()	lookup()	getMessage()	catch()	catch()
32	The _________ is at the center of every remote object because the remote interface defines how the client views the object.	API	remote interface	server program	client program	remote interface
33	RMI handles transmission of requests and provides the facility to load the object's bytecode, which is referred to as __________.	static code loading	dynamic code loading	object code loading	none of the above	dynamic code loading
34	The ________ method registers the remote object with the RMI remote object registry or with another naming service.	rebind()	bind	unbind()	none of the above	rebind()
35	A _________ serves as a firewall and grants or rejects downloaded code access to the local file system and similar privileged operations.	server program	client program	security manager	none of the above	server program
36	Reference to a remote object can be _________.	bound	unbound	rebound	bound, unbound, and rebound	bound, unbound, and rebound
37	A JSP is called by a ___________.	server	client	web service	EJB	client
38	Once a ________ is created, it must be placed in the same directory as HTML pages. the root element of the deployment descriptor.	servlet	JSP	c)EJB	EIS	JSP

39	Once a _________ is created, it must be placed in a particular directory that is included in the CLASSPATH	servlet	JSP	EJB	none of the above	servlet
40	There are ___________factors one must address when installing a JSP.	2	3	4	5	3
41	A JSP program consists of a combination of ___________.	servlets and HTML tags	servlets and EJB tags	HTML tags and JSP tags	servlets and JSP tags	HTML tags and JSP tags
42	288. A powerful feature available in _________ is the ability to change the flow of the program to truly create dynamic content for a web page based on conditions received from the browser.	servlet	JSP	EJB	EIS	JSP
43	The _________ statement in JSP is divided into several JSP tags.-beans> element.	IF	WHILE	DO…WHILE	SWITCH	SWITCH
44	A pair of HTML table data cell tags ________ are placed inside the FOR loop along with a JSP tag that contains an element of the array.	<TB>	<TD>	<TR>	<TC>	<TD>
45	JSP virtual machine runs on a ____________ .	web browser	web server	windows	DOS	web server
46	TOMCAT is one of the most popular JSP ___________.	webbrowser	client program	virtual machine	none of the above	virtual machine
47	Java Beans works on _________.	JDK	BDK	SDK	FDK	BDK

48	The request string sent to the JSP by the browser is divided into ____________ general components that are separated by the question mark.	2	3	4	5	2
49	The secured version of HTTP is __________.	SHTTP	SVHTTP	HTTPS	HTTPSV	HTTPS
50	The __________ enables JSP programs to track multiple sessions simultaneously while maintaining data integrity of each session.	unique password	unique ID	unique username	different username	unique ID
51	__________ attributes can be retrieved and modified each time the JSP program runs.	Servlet	JSP	Session	EJB	Session
52	A session object stores ____________.	implicit data	explicit data	attributes	hidden fields	attributes
53	One of the ________ syntax given below removes a page scope from the stack.	abstract Map peekPageScope ()	abstract Map popPageScope( )	abstract Map pushPageScope().	map push()	abstract Map peekPageScope( )