

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

CS-25 Advanced Java Programming (J2EE)					
SR NO.	Topics	Details	Weightage in %	Lectures	Page No.
1	The J2EE Platform, JDBC (Java Database Connectivity)	<ul style="list-style-type: none"> • Introduction to J2EE • Enterprise Architecture Styles: <ul style="list-style-type: none"> ♣ Two-Tier Architecture ♣ Three-Tier Architecture ♣ N-Tier Architecture • Enterprise Architecture • The J2EE Platform • Introduction to J2EE APIs (Servlet, JSP, EJB, JMS, JavaMail, JSF, JNDI) • Introduction to Containers • Tomcat as a Web Container • Introduction of JDBC • JDBC Architecture • Data types in JDBC • Processing Queries • Database Exception Handling • Discuss types of drivers • JDBC Introduction and Need for JDBC • JDBC Architecture • Types of JDBC Drivers • JDBC API for Database Connectivity (java.sql package) • Statement, PreparedStatement • CallableStatement • ResultSetMetaData • DatabaseMetaData • Other JDBC APIs • Connecting with Databases (MySQL, Access, Oracle) 	20	12	5
2	RMI Servlet	<ul style="list-style-type: none"> • RMI overview • RMI architecture • Stub and Skeleton • Developing and Executing RMI 	20	12	24

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

		application <ul style="list-style-type: none"> • Servlet Introduction • Architecture of a Servlet • Servlet API (Javax.servlet and javax.servlet.http) • Servlet Life Cycle • Developing and Deploying Servlets • Handling Servlet Requests and Responses • Reading Initialization Parameters • Session Tracking Approaches (URL Rewriting, Hidden Form Fields, Cookies, Session API) • Servlet Collaboration • Servlet with JDBC 			
3	JSP, Java Beans	<ul style="list-style-type: none"> • Introduction to JSP and JSP Basics • JSP vs. Servlet • JSP Architecture • Life cycle of JSP • JSP Elements: Directive Elements, Scripting Elements, Action Elements <ul style="list-style-type: none"> ♣ Directive Elements (page, include, taglib) ♣ Scripting Elements (Declaration, scriptlet, expression) ♣ Action Elements (JSP:param, JSP:include, JSP:Forward, JSP:plugin) • JSP Implicit Objects • JSP Scope • Including and Forwarding from JSP Pages • include Action • forward Action • Working with Session & Cookie in JSP • Error Handling and Exception Handling with JSP • JDBC with JSP 	20	12	49

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

		<ul style="list-style-type: none"> • JavaBean Properties • JavaBean Methods • Common JavaBean packaging 			
4	MVC Architecture, EJB, Hibernate	<ul style="list-style-type: none"> • Introduction to MVC • Implementation of MVC Architecture <ul style="list-style-type: none"> • Introduction • Benefits of EJB • Restriction on EJB • Types of EJB • Session Beans • Entity Beans • Message-driven beans • Timer service • Introduction to Hibernate • Need for hibernate • Features of hibernate • Disadvantages of Hibernate • Exploring Hibernate Architecture • Downloading and Configuring and necessary files to Hibernate in Eclipse • Jars files of hibernate. <ul style="list-style-type: none"> • Hibernate Configuration file • Hibernate Mapping file • Basic Example of Hibernate • Annotation • Hibernate Inheritance • Inheritance Annotations • Hibernate Sessions 	20	12	64
5	Spring, Struts	<ul style="list-style-type: none"> • Introduction of Spring Framework • Spring Architecture • Spring Framework definition • Spring & MVC • Spring Context definition • Inversion of Control (IoC) in Spring • Aspect Oriented programming in Spring (AOP) • Understanding Struts Framework 	20	12	93

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

		<ul style="list-style-type: none">• Comparision with MVC using RequestDispatcher and the EL• Struts Flow of Control• Processing Requests with Action Objects• Handling Request Parameters with FormBeans• Prepopulating and Redisplaying Input Forms• Using Properties Files			
--	--	---	--	--	--

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Unit-1 The J2EE Platform, JDBC(Java Database Connectivity)

Introduction to Java 2 Enterprise Edition

Java was developed by James Gosling, an engineer of Sun Microsystems in 1991. Java was previously known as “Oak” but it was an existing language so there was a need to rename it. It was renamed with name “JAVA” in 1995. Java was the Latin name of the coffee beans. Initially it was used to develop consumer based applications such as Microwave, remote controls, TV/VCR and so on. Then it was used with application software based applications. Then after Java Enabled Web browsers were also invented, which are known as Hot Java Browser.

Java 2 platform is divided into three platforms J2SE (java 2 Standard Edition), J2EE(Java2 Enterprise Edition) and J2ME(Java 2 Micro Edition).

Core java application only provides facility to work with windows based applications. So if we required developing the enterprise level application, it will not work. Advance java programming (J2EE) provides the facility to develop the Enterprise based and Internet based application. J2EE also provides a way to develop distributed applications with its APIs.

J2EE is an open source. It reduces the cost of developing Multi-tier applications. It provides a number of APIs to develop the different types of applications, such as JDBC (Java Database Connectivity) used for making connection between the java programs and database. JSP and servlets to create the Dynamic WebPages, EJB for creating the business logic, RMI (Remote Method Invocation) to create the remote objects, client and server based application etc. so it provides distributed computing framework and with this feature it reduces the complexity of the application.

J2EE platform also include concept of containers to provide the interface between the clients, server and the database. It defines a flexible standard that can be built on either a single computer or deployed across several servers, each providing a specific set of J2EE supporting services.

Now a day's J2EE provides many frameworks such as JSF(Java Server Faces), spring, struts, hibernate to develop very attractive, secure, user-friendly, database driven enterprise applications.

INTRODUCTION TO ENTERPRISE APPLICATION DESIGN FRAMEWORK

Enterprise Application Design is divided into 6 logical layers, which is related to logic of the client tier, middle tier and database tier. It defines which layer belongs to which tier.

- **Presentation Manager**

The presentation manager defines the user interface. It always resides on the client tier. It manages the information displayed to the user

- **Presentation Logic**

The presentation logic defines the navigation system of the user interface, how and what will be displayed to the user. It may reside with the client tier or business tier or database tier based on thin client and thick client and application tier.

- **Application Logic**

Application logic defines actual application logic with it. Application logic can be connectivity, validations etc. It may reside with the client tier or business tier or database tier on thin client and thick client and application tier.

- **Business Logic**

Advance Java Programming

The business logic layer contains the business rules of the application. It should be shared with the whole application. It may reside with the business tier or database tier based on thin client and thick client and application tier.

- **Database Logic**

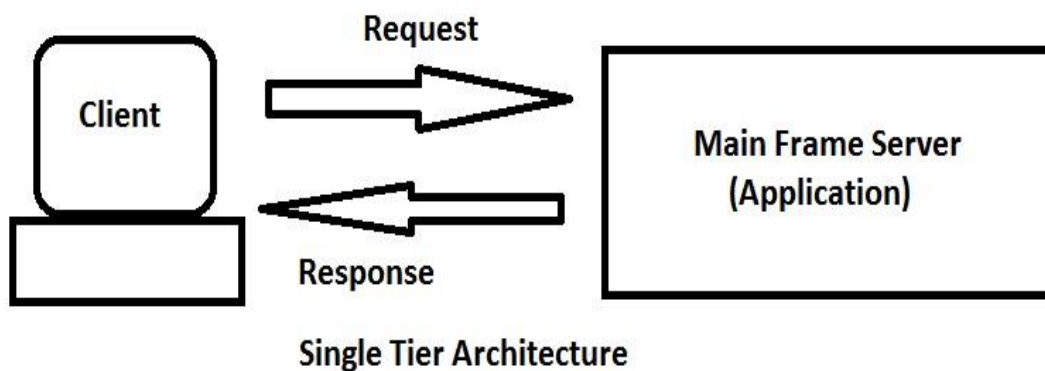
The database logic defines the table structure and the relationship between the tables. It also includes all the constraints of the table. It always resides with database tier.

- **Database Manager**

The Database Manager stores the persistent data. It always resides with database tier.

SINGLE TIER ARCHITECTURE

Simple software applications are written to run on a single computer. User inputs its verification, business logic and data accessed all these could be found together on sing computer, this kind of architecture is called as Single Tier Architecture. Because all logic application services, the presentations, business rules and the data access layers exists in a single computing layer.



- **Advantage**

Single tier system is relatively easy to manage and data consistency is simple because data is stored at only one single location

- **Disadvantage**

However in the world now a days single storage location is not sufficient because of the changing business needs. With the single tier application we cannot share the data in the large amount. It can also not handle multiple users. Because of many such reasons, two tier Architecture is required.

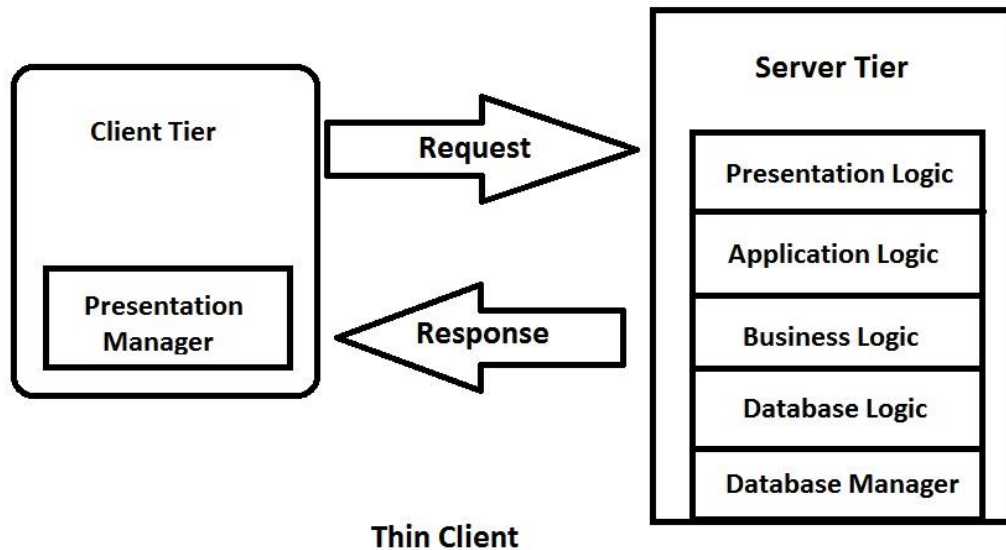
TWO TIER ARCHITECTURE

Application which is divided into two separate tiers, client machine and database server machine is called as **Two Tier Architecture application**. The application includes the presentation and business logic. Data is accessed by connection client machine to a database which is lying on another machine.

Thin Client

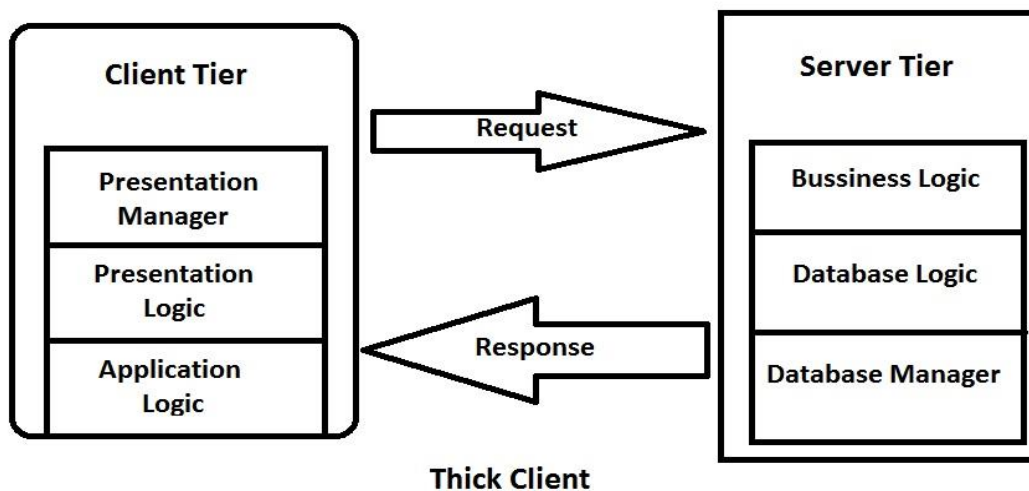
With the two tier architecture, if the presentation manager resides only with the client tier then the client is called as **thin client**. Other presentation logic, application logic, business logic, data logic and database manager reside with the server side (database)

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming



Thick Client

With the two tiers architecture if the presentation manager, presentation logic, application logic reside with the client tier then the client is called as **thick client**. Others like business logic, data logic and database manager reside with the server side (database)



Normal Client

With the two tiers architecture if the presentation manager and presentation logic reside with the client tier then the client is called as **normal client**. Others like application logic, business logic, data logic and database manager reside with the server side (database).

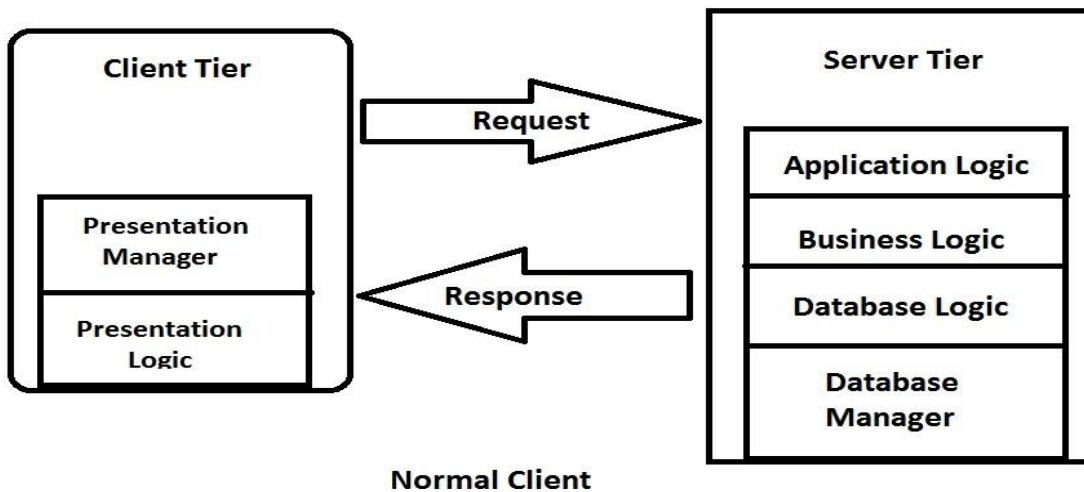
Advantage

Any changes made in data access logic will not affect the presentation and business logic. With the two tier architecture it is easy to develop an application.

Disadvantage

One of the disadvantages of Two Tier Architecture is that the application is expected to support a limited number of users. The reason is that each client requires its own connection and each connection

requires CPU and memory. As the number of connections increases, the database performance degrades



THREE TIER ARCHITECTURE

Application which is divided into three tiers client tier, middle tier and database (Enterprise Information System) tier is known as three tier architecture application. Logic physically separates the business rules. The presentation layer and logic runs on client machine, application and business logic runs on J2EE Server and database logic is there with database layer.

Thin Client

With the three tier architecture if the presentation manager resides only with the client tier then the client is called as **thin client**. Presentation logic, application logic and business logic are with the business tier and database logic and database manager are with the EIS tier (database)

Thick Client

With the three tier architecture if presentation manager, presentation logic, application logic reside with the client tier then the client is called as **Thick client**. Business logic is only with the business tier. The database logic and database manager are with the EIS tier (database)

Advantages

It improves scalability since the application servers can be deployed on many machines. The database no longer requires a connection from every client--- it only requires connections from a smaller number of application servers. It provides better reusability because the same logic can be initiated from many clients or applications. It provides security because client does not have direct access to the database.

Disadvantages

It increases the complexity because to develop the three tier application is more difficult than developing a two tier applications.

N TIER ARCHITECTURE

An application which is divided into more than three tiers can be called N- Tier architecture. In N Tier architecture it is not decided how many tiers can be there, it depends on computing and network hardware on which application is deployed.

Basically it is divided into four application layers: client tier, web tier, business (EJB) tier and database tier.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Client tier

Client tier consists of the user interface for user request and print the response. Client tier runs on client machine. It basically uses the browser or applet as client side applications.

Web tier

Web tier consists of the JSP and servlet dynamic web page to handle the http specify request logons, accesses the business services, and finally constructs a response and send it back to the client.

Business tier

Business tier consists of the business logic for the J2EE application. For example the EJB (Enterprise Java Beans). The benefit of having a centralized business tier is that same business logic can support different types of clients like browser , WAP, other stand-alone application etc.

EIS Tier

EIS tier consists of the DBMS/RDBMS. It handles the users SQL Request and generates appropriate response based on queries. It is responsible for communicating with external resources such as legacy systems, ERP systems, messaging systems like MQSeries etc. It also stores all persistent data in the database.

J2ee is based on N – tier or multi tier architecture applications. J2EE makes easy to develop the Enterprise Application based on 2, 3 or more application layers. Here it also proves that the J2EE is distributing computing framework and multi-tiered application.

Advantages

Separation of user interface logic and business logic is done. Business logic resides on small number of centralized machines. Easy to maintain, to manage, to scale, loosely coupled etc. Additional features can be easily added.

Disadvantages

It is having more complex structures and difficult to setup and maintain all the separated layers.

INTRODUCTION TO WEB SERVER

Web server is basically used to handle the Http request and Http response. In j2ee web server is Apache Http tomcat server to handle the client request and send the response to the client. It also contains the dynamic web pages such as JSP, servlets and XML to handle such requests and responses.

HTTP

A browser can send request and other information. The information can be parameters either by embedding them in the URL or by sending a data stream with the request. This suggests that an HTTP request can be integrated as a database query and query results can be used to build an HTML document dynamically

HTTP consists of set of commands written as lines of ordinary ASCII values.

When we use a web browser, we do not enter HTTP command directly instead when we type URL or click a hyperlink, the browser actions into the HTTP commands that request the document from the server specified in URL.

HTTP as a state less Protocol

HTTP protocol is known as stateless protocol because when client requests to the web server and once server has sent to the client, it forgets clients identity, next time when client again makes a request to the web server at that time web server cannot recognize that client it means web server does not have any information related to client request and response which are made in the past.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

But sometimes it is necessary to keep conversation state with client across the multiple requests. For that we can use some mechanism that store and maintain conversation between client and server. **For ex, HTTP SESSION, COOKIE etc.**

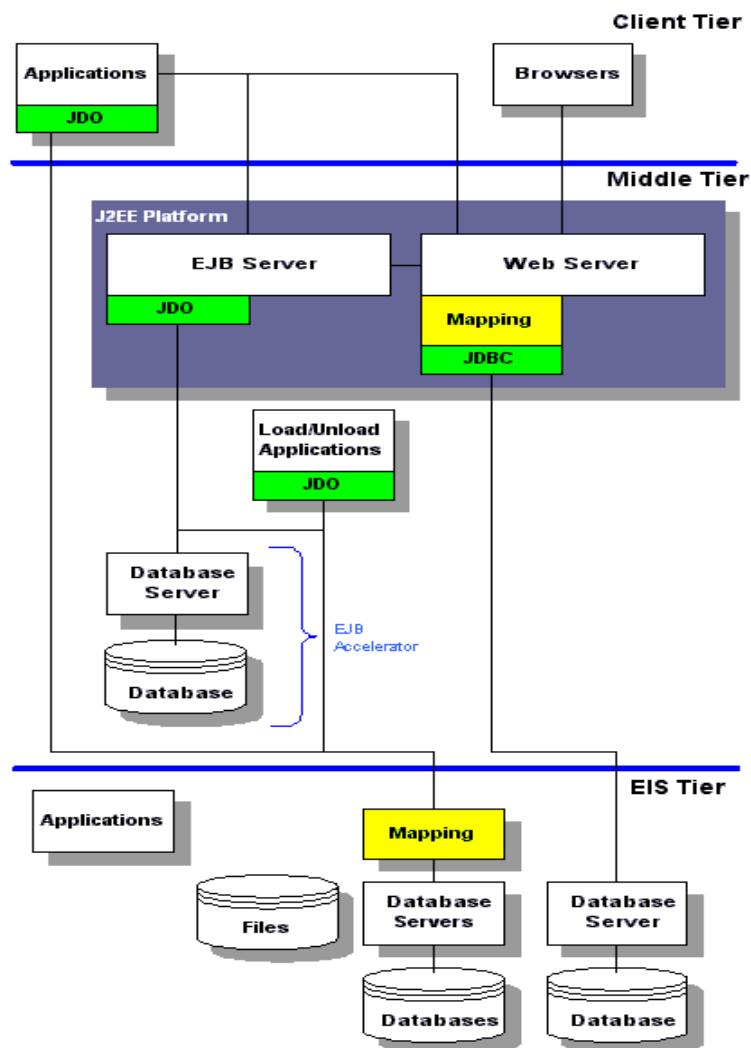
ENTERPRISE ARCHITECTURE OF J2EE

Java to enterprise edition is basically developed for commercial projects and web solutions. Business solutions for commercial project is solved using multitiered architecture. The J2EE platform uses a multi-tiered distributed application model for enterprise applications.

By dividing application logic into the various components according to its task or function and the various application components that are gathered as a J2EE application are installed on different machines depending on the tier in the multi-tiered J2EE environment to which the application component belongs.

Following tiers are available in J2EE:

- (1) Client tier → client machine.
- (2) Web tier → J2EE server.
- (3) Business tier → J2EE server.
- (4) Enterprise information system (EIS)- tier software → EIS server (database)



J2EE Architecture is divided into three or four tier, it is known as Multi-tiered Architecture. But actually those four tiers are divided into the three locations : (1) client Machine (2) J2EE Server (3) EIS Server.

J2EE PLATFORM

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Introduction edition of JAVA.

There are 3 editions of JAVA.

- (1) J2SE: stands for Java 2 standard Edition and is normally used for developing desktop applications.
- (2) J2EE: stands for Java 2 Enterprise edition for applications which run on servers, for example web sites.
- (3) J2ME: stands for Java 2 Micro Edition for applications which run on small scale devices like cell phones, for example games.

What is j2ee?

J2EE (Java 2 enterprise Edition) is an open standard which is provided by sun Microsystems for applications which run on servers. It provides multi-tiered architecture for commercial applications. It includes J2SE + most of the other Java technologies including JavaMail, Activation, Servlets, JSF (Java Server Faces), JMS (Java Messaging Service), EJB (Enterprise Java Beans), and others. Most of the APIs are component-oriented. They are used to provide interfaces for business oriented components for enterprise and distributed internet applications.

J2EE as Multi-tier Architecture

J2EE is a multitier architecture consisting of the client tier, web tier Enterprise Java Beans tier and Enterprise Information System. Two or more tiers can physically reside on the same JVM although each tier provides a different type of functionality as required by the J2EE application. A J2EE application only those tiers whose functionality is required by the J2EE application.

Client tier

The client tier consists of programs that prompt the user for input and then convert into request. The request is then forwarded to software on a component that processes the request and returns results to the client program.

Web tier

The web tier provides Internet functionality to a J2EE applications. Components that operate on the web tier use HTTP to receive requests from and send requests to clients that could reside on any tier.

EJB Tier

The EJB consists of the business logic for J2EE applications. It is here where one or more EJBs reside each encoded with business rules that are called upon indirectly by clients. The EJBs tier is the keystone to every J2EE application because it enables multiple instances of an application to concurrently access business logic and data so as not to slow down the performances.

EIS Tier

The EIS links a J2EE application to resources and legacy systems that are available on the corporate backbone network. It is on the EIS where a J2EE application directly interfaces with DBMS.

Introduction to J2EE APIs

The J2EE platform provides a set of APIs to develop the different types of application. It uses the concept of distributed computing to develop the enterprise application.

▪ Java Database Connectivity (JDBC)

This API is useful to connect our java database application with any relational database. The java application can access data from well know database like, ORACLE, MYSQL, and MSACCESS.

Remote Method Invocation over the internet Inter ORB Protocol (RMI)

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

This API is useful to access objects methods running on different machines.

- **Enterprise Java Beans (EJB)**

This API is useful to define server side components. The J2EE supports components based application development using EJB.

Java Servlets (JS)

The java servlets API provides object oriented abstractions for building dynamic web applications.

Java Server pages (JSP)

The java server pages API provides easy way for building dynamic web applications.

Java Message Service (JMS)

JMS provides a java API for message passing and publish and subscribe the types of message oriented middleware services.

Java Naming and directory Interface (JNDI)

The JNDI API standardizes access to different types of naming and directory services.

Java Transaction API (JTA)

This API is for implementing distributed transaction application.

Java Mail (JM)

This API provides a platform independent and protocol independent frame work to build java based mail application.

Java Server Faces(JSF)

JSF is server-side technology and was built to provide a richer GUI.

An API for representing UI components and managing their state, handling events, Server-side validation, and data conversion, defining page navigation, supporting Internationalization and accessibility, and providing extensibility for all these features.

Introduction To CONTAINERS

There are two types of clients: one is “Thin client” and other is “Thick Client”. Generally thin client in multi-tiered architecture and it is hard to manage. But using component based platform independent architecture of J2EE application because reusable components are stored at business Logic and easily shared within the whole application. For all these components server provides services in the form of containers. Because of these Containers we need not write services for each component included in our application.

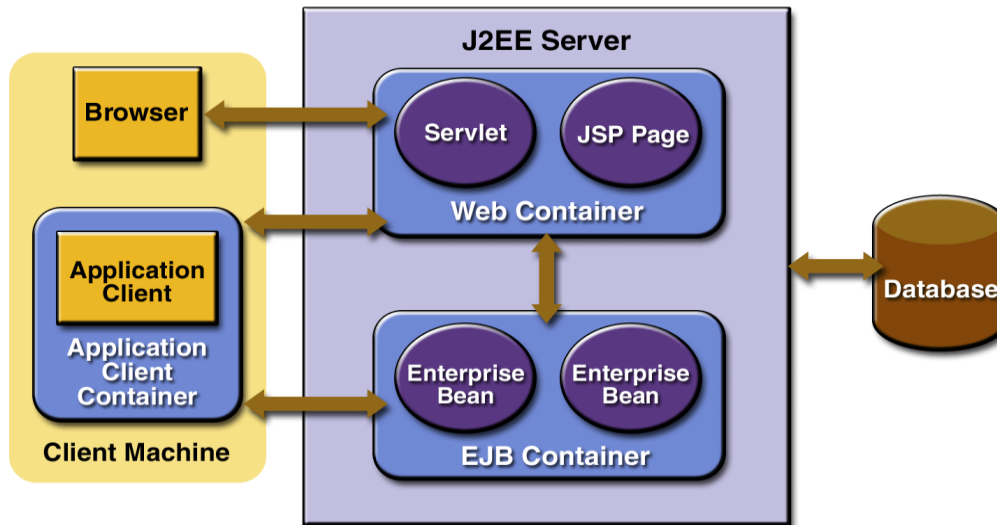
“Containers are interface between a component and client or platform oriented functionality which supports component. Container provides communication platform between client and components.”

Types of container

There are four basic containers in J2EE. In that two server containers and two client containers are listed below.

- (1) Applet Container
- (2) Application Container
- (3) EJB Container
- (4) Web Container

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming



(1) **Applet container** :

Applet container is a client Container which is used to manage an execution of the applet on the browser.

(2) **Application Container** :

Application Container is a Client Container which is used to manage application Client and their components.

(3) **EJB Container**:

EJB container is a server Container which is used to manage Enterprise beans components

(4) **Web Container** :

Web container is also a server Container which is used to manage execution of JSP pages and servlet Components

TOMCAT AS A WEB CONTAINER

Tomcat is a one type of server container, which is used to manage and execute the JSP pages and Servlet components.

Tomcat is a powerful web container provided by Apache. Using tomcat we can develop servlet, JSP application. Tomcat is free web container that can be used to run code of JSP, Servlets. Tomcat is also useful to debug JSP and Servlet pages before deploying on the server. Using JSP and Servlets we can develop dynamic web pages.

Apache is the most common HTTP web server on the intranet which is used at <http://web.njit.edu> tomcat is chosen to be the official sun web components container reference implementation.

- **Basic Requirement**

- Sun Java JDK 5.0 or higher
- Tomcat 6.0 or higher
- Eclipse or netbeans (editor)

- **Class Path**

To set the class path follow following steps :

- Right click on My computer
- Properties-> advanced-> Environment variables -> New

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

- Variable Name=CLASSPATH
- Variable value=c:\tomcat6.0\lib\servletapi.jar;
- Then press OK button

- **Directory structure**

Directory structure of tomcat is as given below :

Bin : all the scripts and batch files are stored into this folder which are used to startup tomcat.

Conf : all the configurations files for global and server configuration and it also contains user authentication settings.

Log : all the server logs resides into this folder.

Lib : JAR files resides into this folder, which is used by tomcat.

Webapps : all the JSP servlets applications folders or directories are stored.

Work : all the temporary and pre- compiled files are stored into this folder.

J2EE 1.4 AS AN APPLICATION SERVER

J2EE technologies and tools are not part of the J2EE 1.4 platform but are provided as a convenience to the developer. The application server includes two user interface technologies Java server pages, Standard tag library and java server tm Faces – that are built on and used in conjunction with the J2EE 1.4 platform technologies Java Servlet and Java Server Pages.

An application server, in N-tier architecture, is a server that hosts an API to expose business logic and business processes for use by third-party application. An application server, a software engine that delivers application to client computer or devices, is required to deploy and built up application across organization. It can handle most, if not all, of the business logic and data access of the application. The web modules such as Servlet and java server pages and business logic are built into J2EE platform.

After the success of the java platform, the term application server sometimes refers to a java platform, enterprise edition (J2EE) application server. J2EE application server is commercial open source application server. Programming language used in application server is java web modules are servlets and java server pages, and business logic is built into enterprise JavaBeans (EJB)

The web Container

The web container is a J2EE container that is used to host web applications, so, the web application runs within a web container of a web server. The web container is used to extend the server functionality, because it provides development environment to the developers to run java server pages and servlet components. It also provides naming context and cycle management using components for runtime environment.

Some of the web servers may also provide additional services such as security and concurrency control. A web server may work with an EJB server to provide some of those services. A web server, however, does not need to be located on the same machine as an EJB server.

The EJB container

Enterprise java beans (EJB) components are components of java programming language server which is used to hold business logic. So, the EJB container provides Local and Remote Access to enterprise Beans.

Types of enterprise beans

- (1) **Entity beans** : it is used for the representation of persistent data which are maintained in a database.

Advance Java Programming

- (2) **Message Driven Beans** : Message Driven Beans are used to convey message to application modules and services in asynchronous manner.
- (3) **Session Beans** : it is used to handle objects and processes which are used by a single client. It exists only for the duration of a single client/server session. It also performs operations such as accessing a database or performing calculations.

Responsibility of EJB Container

- Creating enterprise Bean
- Binding the Enterprise Bean
- Naming Service: So the other components can access the enterprise bean easily.
- Provides Authorization so that authorized clients can do the following :
 - >access the enterprise Bean's methods
 - >Saving the Bean's state to persistent storage
 - >Caching the state of the bean
 - >Activating the bean

Introduction and Need for JDBC

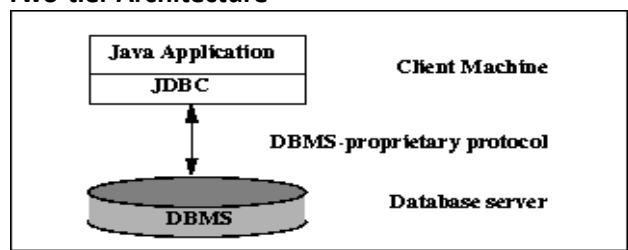
JDBC (Java Database Connectivity) is a standard API for accessing relational databases from a Java program. This interface makes it easy to access a database because it provides an abstract layer that hides the low-level details, such as managing sockets. It also provides for interoperability and portability since it allows a single application to access multiple database management systems simultaneously.

For example, a single application can query and manipulate a database in Oracle and a database in DB2. Communication with a database management system (DBMS) is through method calls. These calls are passed to the driver, which in turn, translates them into DBMS-specific calls. The driver basically acts like a set of library routines. Therefore, to get your program to communicate with a particular DBMS, you need a compatible JDBC driver. The basic steps to get your program up and running are:

1. Load the driver and register it with the driver manager
2. Connect to a database
3. Create a statement
4. Execute a query and retrieve the results, or make changes to the database
5. Disconnect from the database

Steps 1 and 2 are the only DBMS-specific steps. The rest is DBMS independent with one exception: the mappings between the DBMS and Java datatypes is somewhat DBMS-specific. However, this is not a major issue because the driver usually handles the datatype conversions. Therefore, to make your program work with DB2 instead Oracle, you usually only have to change the code associated with steps 1 and 2. As you can see, JDBC is very powerful and flexible.

Two-tier Architecture



Smt J.J.Kundalia Commerce College Department Of Computer Science

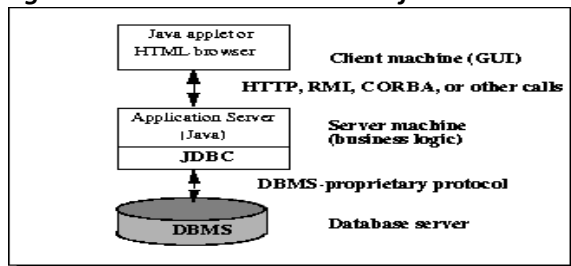
Advance Java Programming

In the two-tier model, a Java 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.

Three-tier Architecture

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.

Figure 2: Three-tier Architecture for Data Access.



JDBC Driver

JDBC drivers implement the defined interfaces in the JDBC API for interacting with your database server. For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The *Java.sql* package that ships with JDK contains various classes with their behaviours defined and their actual implementations are done in third-party drivers. Third party vendors implement the *java.sql.Driver* interface in their database driver.

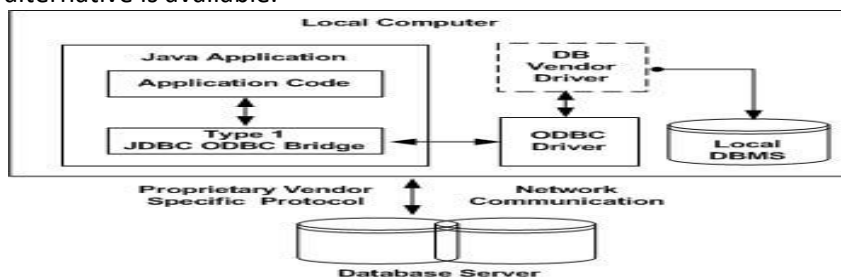
JDBC Drivers Types:

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below:

Type 1: JDBC-ODBC Bridge Driver:

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



Smt J.J.Kundalia Commerce College Department Of Computer Science

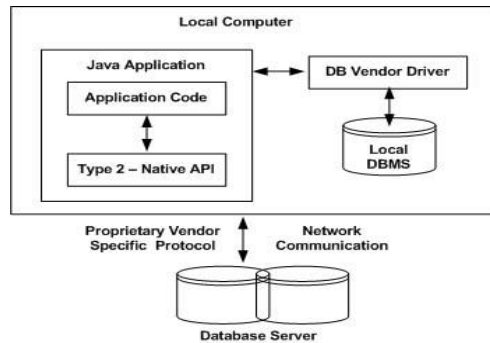
Advance Java Programming

The JDBC-ODBC bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API:

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls which are unique to the database. These drivers typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge, the vendor-specific driver must be installed on each client machine.

If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

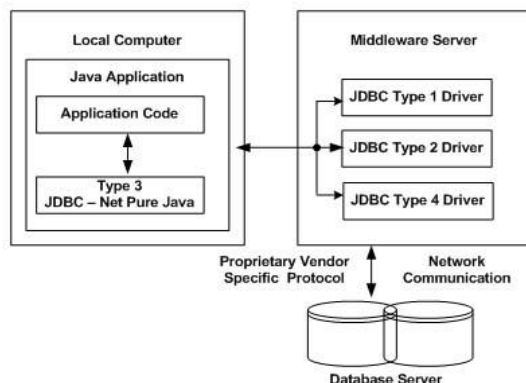


The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java:

In a Type 3 driver, a three-tier approach is used to accessing databases. The JDBC clients use standard network sockets to communicate with an middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

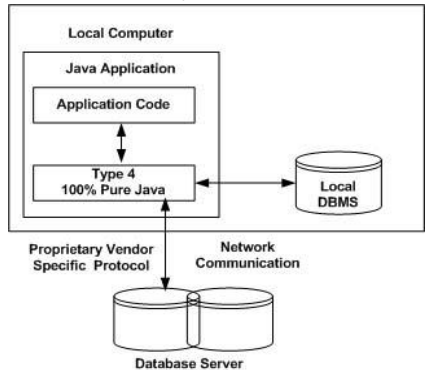
Type 4: 100% pure Java:

In a Type 4 driver, a pure Java-based driver that communicates directly with vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Establishing Connections

First, establish a connection with the data source you want to use. A data source can be a DBMS, a legacy file system, or some other source of data with a corresponding JDBC driver. This connection is represented by a Connection object. See [Establishing a Connection](#) for more information.

STEPS TO CONNECT WITH DATABASE

There are seven standard steps in querying databases:

1. Load the JDBC driver.
2. Establish the connection.
3. Create a statement object.
4. Execute a query or update.
5. Process the results.
6. Close the connection.

1. Load the JDBC driver

Loading the driver or drivers we want to use is very simple and involves just one line of code. If, for example, we want to use the JDBC–ODBC Bridge driver, the following code will load it:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2. Establish the connection

The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

```
Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
```

This step is also simple, the hardest part being what to supply for url. If we are using the JDBC–ODBC Bridge driver, the JDBC URL will start with jdbc:odbc:. The rest of the URL is generally our data source name or database system. So, if we are using ODBC to access an ODBC data source called "Fred", for example, our JDBC URL could be jdbc:odbc:Fred. In place of "myLogin" we put the name we use to log in to the DBMS; in place of "myPassword" we put our password for the DBMS. So if we log in to our DBMS with a login name of "Fernanda" and a password of "J8", just these two lines of code will establish a connection:

```
String url = "jdbc:odbc:Fred";
```

```
Connection con = DriverManager.getConnection(url, "Fernanda", "J8");
```

3. Create a statement object

A Statement object is used to send queries and commands to the database and is created from the Connection as follows:

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

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

4. Execute a query or update

Once we have a Statement object, we can use it to send SQL queries by using the executeQuery method, which returns an object of type Result-Set. Here is an example:

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

To modify the database, use executeUpdate instead of executeQuery, and supply a string that uses UPDATE, INSERT, or DELETE. Other useful method in the Statement class include execute(executeanarbitrarcommand).

5. Process the results

The simplest way to handle the results is to process them one row at a time, using the ResultSet's next method to move through the table a row at a time. Within a row, ResultSet provides various getXxx methods that take a column index or column name as an argument and return the result as a variety of different Java types. For instance, use getInt if the value should be an integer, getString for a String, and so on for most other data types. If we just want to display the results, we can use getString regardless of the actual column type.

However, if we use the version that takes a column index, note that columns are indexed starting at 1 (following the SQL convention), not at 0 as with arrays, vectors, and most other data structures in the Java programming language.

Here is an example that prints the values of the first three columns in all rows of a ResultSet.

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

6. Close the connection

To close the connection explicitly, we would do:con.close();

Creating Statements

A Statement is an interface that represents a SQL statement. You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set. You need a Connection object to create a Statement object.

For example, CoffeesTables.viewTable creates a Statement object with the following code:

```
stmt = con.createStatement();
```

There are three different kinds of statements:

- Statement: Used to implement simple SQL statements with no parameters.
- PreparedStatement: (Extends Statement.) Used for precompiling SQL statements that might contain input parameters. See [Using Prepared Statements](#) for more information.
- CallableStatement: (Extends PreparedStatement.) Used to execute stored procedures that may contain both input and output parameters. See [Stored Procedures](#) for more information.

Executing Queries

To execute a query, call an execute method from Statement such as the following:

execute: Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling statement.getResultSet.

executeQuery: Returns one ResultSet object.

executeUpdate: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using INSERT, DELETE, or UPDATE SQL statements.

For example, CoffeesTables.viewTable executed a Statement object with the following code:

```
ResultSet rs = stmt.executeQuery(query);
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Processing ResultSet Objects

You access the data in a ResultSet object through a cursor. Note that this cursor is not a database cursor. This cursor is a pointer that points to one row of data in the ResultSet object. Initially, the cursor is positioned before the first row. You call various methods defined in the ResultSet object to move the cursor.

For example, CoffeesTables.viewTable repeatedly calls the method ResultSet.next to move the cursor forward by one row. Every time it calls next, the method outputs the data in the row where the cursor is currently positioned:

```
try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
        String coffeeName = rs.getString("COF_NAME");
        int supplierID = rs.getInt("SUP_ID");
        float price = rs.getFloat("PRICE");
        int sales = rs.getInt("SALES");
        int total = rs.getInt("TOTAL");
        System.out.println(coffeeName + "\t" + supplierID + "\t" + price + "\t" + sales +
            "\t" + total);
    }
}
// ...
```

Complete program :--

```
import java.sql.*;
public class jdbc_type1 {
    public static void main(String a[]) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:TEST");
        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery("select * from test");
        while(rs.next()){
            System.out.println(rs.getString(1));
        }
        rs.close();
        st.close();
        con.close();
    }
}
```

1) Prepared Statement

This JDBC tutorial helps us to use the **PreparedStatement** interface of **java.sql** package twice in a program. According to our requirement we can use the **PreparedStatement** object. The **PreparedStatement** object represents the precompiled SQL statement. Whenever, the SQL statement is precompiled then it stored in the **PreparedStatement** object which executes the statement many times and it reduces the time duration of execution. The **PreparedStatement** uses the '?' with SQL statement that provides the facility for setting an appropriate conditions in it. See brief description below:

Instances of [PreparedStatement](#) contain an SQL statement that has already been compiled. This is what makes a statement "prepared" Because [PreparedStatement](#) objects are precompiled, their

Advance Java Programming

execution can be faster than that of Statement objects.
The prepared statement is used to execute sql queries

2)CallableStatement

A [CallableStatement](#) object provides a way to call stored procedures in a standard way for all RDBMSs. A stored procedure is stored in a database; the call to the stored procedure is what a [CallableStatement](#) object contains.

Creating a CallableStatement

You create an instance of a CallableStatement by calling the prepareCall() method on a connection object. Here is an example:

```
CallableStatement callableStatement =  
connection.prepareCall("{call calculateStatistics(?, ?)}");
```

If the stored procedure returns a ResultSet, and you need a non-default ResultSet (e.g. with different holdability, concurrency etc. characteristics), you will need to specify these characteristics already when creating the CallableStatement. Here is an example:

```
CallableStatement callableStatement =  
connection.prepareCall("{call calculateStatistics(?, ?)}",  
    ResultSet.TYPE_FORWARD_ONLY,  
    ResultSet.CONCUR_READ_ONLY,  
    ResultSet.CLOSE_CURSORS_OVER_COMMIT );
```

Setting Parameter Values

Once created, a CallableStatement is very similar to a PreparedStatement. For instance, you can set parameters into the SQL, at the places where you put a ?. Here is an example:

```
CallableStatement callableStatement =  
connection.prepareCall("{call calculateStatistics(?, ?)}");  
callableStatement.setString(1, "param1");  
callableStatement.setInt (2, 123);
```

Executing the CallableStatement

Once you have set the parameter values you need to set, you are ready to execute the CallableStatement. Here is how that is done:

```
ResultSet result = callableStatement.executeQuery();
```

The executeQuery() method is used if the stored procedure returns a ResultSet.

If the stored procedure just updates the database, you can call the executeUpdate() method instead, like this: callableStatement.executeUpdate();

CALLABLESTATEMENT

A CallableStatement object provides a way to call stored procedures in a standard way for all RDBMS. A stored procedure is stored in a database; the call to the stored procedure is what a CallableStatement object contains. This call is written in an escape syntax that may take one of two forms: one form with a result parameter and the other without one. A result parameter, a kind of OUT parameter, is the return value for the stored procedure. Both forms may have a variable number of parameters used for input (IN parameters), output (OUT parameters), or both (INOUT parameters). A question mark (?) serves as a placeholder for a parameter.

The syntax for invoking a stored procedure using the JDBC API is shown here.

=> Calling a stored procedure with no parameters:

=> {call procedure_name}

=> Calling a stored procedure with one or more IN, OUT, or INOUT parameters:

=> {call procedure_name(?, ?, ...)}

=> Calling a procedure that returns a result parameter and may or may not take any IN, OUT, or INOUT parameters: (Note that the square brackets indicate that what is between them is optional; they are not

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

themselves part of the syntax.)

```
=> {? = call procedure_name[?, ?, ...]}
```

IN Parameters

Passing in any IN parameter values to a CallableStatement object is done using setter methods. These methods include both the setter methods inherited from the PreparedStatement interface and those defined in the CallableStatement interface. The type of the value being passed in determines which setter method to use (setFloat to pass in a float value, setBoolean to pass in a boolean, and so on).

The following code fragment uses the setter methods that take the parameter number to indicate which parameter is to be set.

```
String sql = "{call updateStats(?, ?)}";
CallableStatement cstmt = con.prepareCall(sql);
cstmt.setInt(1, 398); cstmt.setDouble(2, 0.04395);
```

OUT Parameters

If the stored procedure returns OUT parameters, the data type of each OUT parameter must be registered before the CallableStatement object can be executed. This is necessary because some DBMSs require the SQL type (which the JDBC type represents); the JDBC API itself does not require that the SQL type be registered. JDBC types, a set of generic SQL type identifiers that represent the most commonly used SQL types.

Registering the JDBC type is done with the method registerOutParameter. Then after the statement has been executed, the CallableStatement interface's getter methods can be used to retrieve OUT parameter values. The correct getter method to use is the type in the Java programming language that corresponds to the JDBC type registered for that parameter.

```
CallableStatement cstmt = con.prepareCall("{call getTestData(?, ?)}");
cstmt.registerOutParameter(1, java.sql.Types.TINYINT);
cstmt.registerOutParameter(2, java.sql.Types.DECIMAL);
ResultSet rs = cstmt.executeQuery();
```

INOUT Parameters

A parameter that supplies input as well as accepts output (an INOUT parameter) requires a call to the appropriate setter method in addition to a call to the method registerOutParameter.

The setter method sets a parameter's value as an input parameter, and the method registerOutParameter registers its JDBC type as an output parameter. The setter method provides a Java value that the driver converts to a JDBC value before sending it to the database.

The JDBC type of this IN value and the JDBC type supplied to the method registerOutParameter should be the same. If they are not the same, they should at least be types that are compatible, that is, types that can be mapped to each other. Then, to retrieve the output value, a corresponding getter method is used.

```
CallableStatement cstmt = con.prepareCall("{call reviseTotal(?) }");
cstmt.setByte(1, (byte)25);
cstmt.registerOutParameter(1, java.sql.Types.TINYINT);
cstmt.executeUpdate();
byte x = cstmt.getByte(1);
```

for example

```
import java.sql.*;
public class calldemo {
    public static void main(String a[]) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:TEST");
```

Advance Java Programming

```
CallableStatement ps=con.prepareCall("{ call pdemo(?,?)}");
ps.setInt(1,2);
ps.registerOutParameter(2,java.sql.Types.VARCHAR);
ps.executeUpdate();
String s=ps.getString(2);
System.out.println(s);
} }
```

MetaData

The definition of *metadata* is "data about other data." With a file system, the data is contained in its files and directories, and the metadata tracks information about each of these objects: Is it a regular file, a directory, or a link? What is its size, creation date, last modified date, file owner, group owner, and access permissions?

Most JDBC programs are designed to work with a specific database and particular tables in that database; the program knows exactly what kind of data it is dealing with. Some applications, however, need to dynamically discover information about result set structures or underlying database configurations. This information is called *metadata*, and JDBC provides two classes for dealing with it: `DatabaseMetaData` and `ResultSetMetaData`. If you are developing a JDBC application that will be deployed outside a known environment, you need to be familiar with these interfaces.

DatabaseMetaData

You can retrieve general information about the structure of a database with the `java.sql.DatabaseMetaData` interface. By making thorough use of this class, a program can tailor its SQL and use of JDBC on the fly, to accommodate different levels of database and JDBC driver support. Database metadata is associated with a particular connection, so `DatabaseMetaData` objects are created with the `getMetaData()` method of `Connection`:

```
DatabaseMetaData dbmeta = con.getMetaData();
```

The key methods used here are `getColumnCount()`, `getColumnLabel()`, and `getColumnTypeName()`. Note that type names returned by `getColumnTypeName()` are database-specific

The `ResultSetMetaData` interface provides information about the structure of a particular `ResultSet`. Data provided by `ResultSetMetaData` includes the number of available columns, the names of those columns, and the kind of data available in each

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Unit-2 RMI ,Servlet

We know that in client/server system client makes a request to the server and the server fulfills its request and provides appropriate response to the client. There are three components of client/server system.

- (1) Presentation Layer (Information Layer)
- (2) Information Processing Layer
- (3) Information Storage Layer,

Presentation layer is implemented by client system sometimes it is also known as web browser. But the Information Processing Layer is implemented either by Client or Server or Server Support System and it depends upon the architecture of the application. The information storage Layer is implemented on web Server or FTP server or J2EE server.

According to the behavior of the application Information Processing Layer may be at Client side or it may be at Server side. We have already discussed concept of thin client and thick client. In thin client processing layer resides at server side. And in the thick client processing layer resides at Client side.

In distributed application, following different types of method invocation are used.

- (1) Distributed Computing Environment (DCE).
- (2) Distributed Components Object Model (DCOM).
- (3) Common Object Request Broker Architecture (CORBA)
- (4) Java's Remote Methods Invocation (RMI)

DISTRIBUTED COMPUTING ENVIRONMENT (DCE)

Distributed computing environment is an industry standard for the open system foundation (OSF). It is vendor-neutral set of distributed computing. It provides security and control for accessing data. In distributed environment all the components that provides individual functionality is called DCE cell. For ex, a college system is divided into different departments like chemistry, biology, Mathematics, Computer science, admin etc.

In each DCE cell many services and technologies are used which are described as follow:

- Directory services
- Distributed file Services (DFS)
- Distributed time Services (DTS)
- Security Services
- Remote Procedure Call (RPC)
- DCE Threads

DISTRIBUTED COMPONENTS OBJECT MODEL (DCOM)

Distributed Component Object Model (DCOM) is a proprietary Microsoft technology for communication among software components distributed across networked computers. DCOM, which originally was called "Network OLE", extends Microsoft's COM, and provides the communication substrate under Microsoft's COM+ application server infrastructure. It has been deprecated in favor of the Microsoft .NET Framework.

The addition of the "D" to COM was due to extensive use of DCE/RPC (Distributed Computing Environment/Remote Procedure Calls) – more specifically Microsoft's enhanced version, known as MSRPC.

In terms of the extensions it added to COM, DCOM had to solve the problems of

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

- Marshalling – serializing and deserializing the arguments and return values of method calls "over the wire".
- Distributed garbage collection – ensuring that references held by clients of interfaces are released when, for example, the client process crashed, or the network connection was lost.

One of the key factors in solving these problems is the use of DCE/RPC as the underlying RPC mechanism behind DCOM. DCE/RPC has strictly defined rules regarding marshalling and who is responsible for freeing memory.

DCOM was a major competitor to CORBA. Proponents of both of these technologies saw them as one day becoming the model for code and service-reuse over the Internet. However, the difficulties involved in getting either of these technologies to work over Internet firewalls, and on unknown and insecure machines, meant that normal HTTP requests in combination with web browsers won out over both of them. Microsoft, at one point, attempted and failed to head this off by adding an extra http transport to DCE/RPC called ncacn_http (Network Computing Architecture, Connection-based, over HTTP)

COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

CORBA is a most popular architecture for building, distributed application. Using CORBA we can develop object oriented distributed applications. Like other Architecture CORBA is not proprietary architecture. It is an open standard and objects are accessed with ORBs (Object Request Brokers). Client interface of ORB is known as IDL (Interface Definition Language) Stub and server interface is known as IDL Skeleton. Advantages of CORBA are that due to open standard it works on all platforms. Clients and servers can be distributed and language independent. So, we have to provide only IDL interface to object.

JAVA REMOTE METHODS INVOCATION (RMI)

RMI is for Java's Distributed Object Model. So, friends your big question might be that why we need RMI though we have DCE, DCOM AND CORBA?

And here is your answer: Remote Procedure Call (RPC) of DCE is not suitable with Java's Object Oriented Distributed Application (OODA). DCOM has good performance with windows because it is a proprietary of Microsoft so, DCOM does not support java object fully. CORBA is excellent but java Programmer will have to learn IDL (Interface definition Language) to develop Java Distributed application (JDA).

All these methods have their own advantages and disadvantages in terms of complexity, performance and cross-platform compatibility. But RMI is purely java-specific which provides Java to java communications only and as a result, RMI is much simpler than CORBA.

Introduction to RMI

As we discussed before that DCE, DCOM DCOM and CORBA are not compatible for J2EE environment. So, RMI is invented for J2EE distributed application. J2EE provides the distributed programming environment for the users. It is having number of APIs to develop Enterprise Application. RMI is the one of the API for developing the distributed Application.

Distributed application contains the distributed objects over the network. It communication with each other by using transport protocol or basically using TCP/IP protocol. Client sends the request to the server to access the Remote objects and server sends the response related to client Request.

This kind of client-server computing is possible with java by using RMI (Remote Methods Invocation API). Client can access the Remote objects by invoking the Methods. Lets us discuss exact definition of RMI in the concept of java as programming language.

Advance Java Programming

“RMI is the technology which allows the client to remote object communication object to object communication between different java virtual machine. “ In short, RMI provide remote communication between programs written in the java programming language.

An overview of RMI Application

RMI applications basically divide into programs client and server. A server program creates some remote objects and makes the references to them and waits for the client request to invoke methods on the remote objects.

A client application gets a remote reference to one or more Remote objects from the server and invokes the methods on them. RMI provides the mechanism by which the server and the client communication and pass information back and forth. Such application is to say a Distributed Object application (DOA).

RMI Architecture

RMI architecture defines the how the client request to the server for the remote objects and how the server processes for that request. It also defines the implementation of remote class on a remote JVM. RMI architecture basically divides in four layers:

1. Application Layer
2. Proxy Layer(stud and Skeleton)
3. Remote reference Layer
4. Transport Layer

Each layer can perform specific functions, like establish the connection, assemble and disassemble the parameters, transmitting the object etc. All above RMI layer are independent from each other and they can be used based on the client request for the service as shown in fig. the garbage collection is one of the advantage of the java language. RMI also supports garbage collection. The registry simply keeps track of the addresses of remote objects that are being exported by the applications. All the distributed systems use the registry to keep track of the names of the remote objects.

1. Application layer

Application layer consists of the client side java program and the server side java program with the remote methods. Here, the high level calls are made in order to access and explain remote objects. The client can access the remote method through an interface that extends java.rmi.Remote. When we want to define a set of methods that will be remotely called they must be declared in one or more interface that should extend java.rmi.Remote.

Once the methods described in the remote interface have been implemented, the object must be exported using UnicastRemoteObject class of the java.rmi.server package. Then the application will register itself with registry and it is used by client to obtain reference of the objects. On the client side a client simply request a remote object from either registry or remote object whose reference has already been obtained. After getting reference the proxy layer has to perform the important role of communication between the client and server. In our application, the RemoteMessageClient and RemoteMessageServer from the application layer.

2. Proxy Layer

The stub/skeleton layer is responsible for listening to the remote method calls made by the client and redirecting them to the server. This layer consists of a stub and a skeleton. The stub and skeleton are created using the RMI compiler (RMIC). These are simply class files that represent the client and server side of a remote object. A stub is a client side proxy representing the remote objects. The stub

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

communicates the method invocations to the remote object through a skeleton that is implemented on the server. The skeleton is a sever-side proxy that communicates with the stub. It makes a call to the remote object. Therefore it is also known as proxy layer. It is also interface between application layer and all RMI layers.

3. Remote Reference Layer

Remote reference layer is interface between the proxy layer and transport layer, it handles actual communication protocols. The Remote Reference Layer (RRL) interprets the references made by the client to the remote object on the server. This layer is present on the client as well as on the server. The RRL on the client-side receives the request for the methods from the stub. The request is then transferred to the RRL on the server-side. When transmitting the parameters or objects through the network it should be in the form of a stream. The java virtual Machine (JVM) works with the java Byte codes. It can get the stream-oriented data from the transport layer and give it to the proxy layer and vice versa.

RRL is used to:

- Handle the replicated objects. Once the feature is incorporated into the RMI system, the replicated objects will allow simple dispatch to other programs.
- It is responsible for establishing persistence and strategies for recovery of lost connections.

4. Transport layer

Transport layer manages the connection between client remote reference layer and remote server machine. It receives a request from the client-side RRL and establishes a connection with the server through server-side RRL.

The transport layer has following responsibilities:

- It is responsible for handling the actual machine to machine communication the default communication will take place through a standard TCP/IP.
- It creates a stream that is accessed by the remote reference layer to send and receive data to and from other machines.
- It sets up the connection to remote machines.
- It manages the connections.
- It monitors the connection to make sure that they (remote machines) are lives.
- It listens for connection from the machines.

STUBS AND SKELETONS

The stub class

The stub is a client side proxy of the remote object. It has three primary responsibilities.

1. It presents the same remote interfaces as the object of server (ex. RemoteMessageClient), the stub is equivalent to the remote object.
2. It works with java virtual machine (JVM) and RMI System on client machine to serialize any arguments to a remote method call and sends this information to server machine.
3. The stub receives any result from the remote method and returns it to the client.

The skeleton class

The skeleton is a server side proxy of the remote objects. It has three primary responsibilities.

1. It receives the remote method call and any associated arguments. It works with the JVM and RMI system on the sever machine to deserialize any arguments for the remote method call.
2. It invokes the appropriate method in the server (Ex.RemoteMessageServer) using arguments.

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

3. It receives any return value from this method call and works with the JVM and RMI system on server machine to serialize this return value and sends the information back to client (Ex. RemoteMessageClient).

To develop an RMI application, we need to create the following java programs :

1. Remote Interface
2. Remote object class
3. Server program
4. Client program

To develop the following java files we need to use java.rmi.* (package) and java.rmi.server package (API)

1) Create Remote Interface

```
import java.rmi.*;
public interface rmiInterface extends Remote
{
    double add (double d1, double d2) throws Exception;
}
```

2) Create Remote object class.

```
import java.rmi.*;
import java.rmi.server.*;
public class rmiImpl extends UnicastRemoteObject
    implements rmiInterface
{
    public rmiImpl () throws RemoteException
    { }
    public double add (double d1, double d2) throws Exception
    {
        return d1+d2;
    }
}
```

3) it is to code a driver program, which will create an instance of the class, and register it with the rmiregistry

```
import java.rmi.*;
import java.rmi.server.*;
public class RemoteMsgserver
{
    public static void main (String args [])
    {
        try
        {
            rmiImpl obj=new rmiImpl ();
            Naming.rebind ("///rmi", rmiobj);    }
        catch (Exception e) { System.out.println ("An error occurred trying to "+
            "bind the object to the registry.");
        }
    }
}
```

4)Create implement the client.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
import java.rmi.*;
import java.io.*;
import java.rmi.server.*;
public class ConfusedClient
{
    public static void main (String args [])
    {
        BufferedReader reader;
        try
        {
            String url="rmi: //" +args [0] +"/rmi";
           .rmiInterface.rmiinter= (rmiInterface) Naming. lookup(url));
            System.out.println ("\n The first no is"+args[1]);
            System.out.println ("\n The second no is"+args[2]);
            double d1, d2;
            d1=Double.valueOf (args [1]).doubleValue ();
            d2=Double.valueOf (args [2]).doubleValue ();
            System.out.println ("\n The sum of two values"+rmiinter.add (d1, d2));
        }
        catch (Exception e)
        {
            System.out.println ("An error occurred.");
        }
    }
}
```

The compiling & running steps:

1. Compile all of the .java files
2. Run rmic on the implementation of the Remote interface
3. Open a second terminal window
4. Type "start rmiregistry" in one of the windows
5. Register the remote object in the registry by running the driver
6. Run the client

Introduction to Servlets

Servlet technology was the original Java based solution for web development however due to the problems of maintaining the HTML within the Java code they were never a great success.

The Servlet Lifecycle

Servlet containers are responsible for handling requests, passing that request to the Servlet and then returning the response to the client. The actual implementation of the container will vary from program to program but the interface between Containers and Servlets is defined by the Servlet API much in the same way as there are many JSP Containers all adhering to the JSP Specification.

The basic lifecycle of a Servlet is,

- The Servlet Container creates an instance of the Servlet.
- The Container calls the instance's init() method.
- If the Container has a request for the Servlet it will call the instance's service() method.
- Before the instance is destroyed the Container will call the destroy() method.
- Finally the instance is destroyed and marked for garbage collection.

Advance Java Programming

Typically the `init()` method is only called once and then the `service()` method is called repeatedly for each request. This is much more efficient than executing `inti()`, `service()`, `destroy()` for each request. What happens, you may ask yourself, when a `service()` method is still executing when the Container receives another request ? Typically this will involve the creation of another program execution thread. In practice, Servlet Containers create a pool of threads to which incoming requests are generally allocated.

A Servlet is defined by the `javax.servlet.Servlet` interface. There is also a `GenericServlet` abstract class which provides a basic implementation of the Servlet interface. However for this discussion we will only look at the `HttpServlet` class which extends the `GenericServlet` class. When you come to write a Servlet it will be this class that you are most likely to extend.

The `service()` method

The `service()` method is implemented by the `HttpServlet` as a dispatcher of HTTP Requests and therefore should never be overridden. When a request is made the `service()` method will determine the type of request (GET, POST, etc) and dispatch it to the appropriate method (`doGet()`, `doPost()`, etc). For the most part you will be overriding `doGet()` and the `doPost()` methods. These have message signatures similar to,

protected void `doXxx(HttpServletRequest request, HttpServletResponse response)`
throws `ServletException`, `java.io.IOException`

We already know the two objects `HttpServletRequest` and `HttpServletResponse` as these are just the implicit objects request and response and can be manipulated in exactly the same way within the body of the `doGet()`, `doPost()` methods.

The `init()` method

The `init()` method is executed when the Servlet is first instantiated. The Servlet container will pass an object of type `ServletConfig` to the method so that container specific configuration data can be stored by that Servlet instance for later use.

Not every Servlet requires that the `init()` method do something. The type of work carried out by this method can include such activities as initiating database connections, establishing default values or instantiating JavaBeans. If your Servlet doesn't require any kind of initialization activity then don't include an `init()` method in your Servlet

The `destroy()` method

You can assume that at any given time the Servlet container will decide to remove the Servlet. This might occur if the Container needs to free some memory or the Servlet hasn't been requested for some time. The `destroy()` method is called prior to removing of the Servlet so you can use this method for any clean up activity that may be required; releasing database connections etc.

There are four parts to this workshop and indeed in writing Servlets. These are,

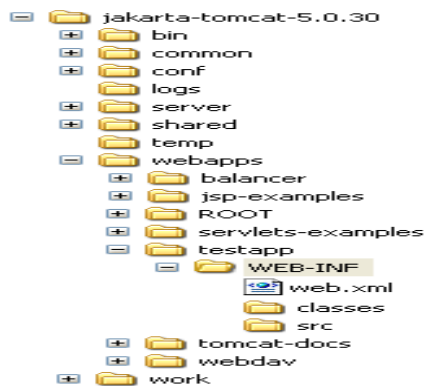
- 1) Write the Servlet
- 2) Define your environment
- 3) Compile the Servlet
- 4) Test the Servlet

1. Write the Servlet

Application Servers can vary in the way they handle Servlets. This workshop assumes you are using Tomcat but if you are using a different Application Server then consult its documentation for details on deploying Servlets.

Advance Java Programming

First of all let us create a new web application with the appropriate folder structure. In Tomcat you'll find a webapps folder below the installation folder. In this folder create a new subfolder called 'testapp' and a subfolder of this called WEB-INF. Below WEB-INF create two subfolders 'classes' and 'src' and a xml file 'web.xml'. Your folder structure should be similar to the following



1)A Servlet is a Java class so let us begin by creating a new text document called Simple.java in the 'src' subfolder of the WEB-INF folder.

The Servlet begins by importing the packages required for the class:

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;
```

Next we define our Servlet class as extending the HttpServlet class,

```
public class Simple extends HttpServlet  
{
```

We will only be overriding the doGet() method so add the following to the body of the Servlet,

```
public void doGet(HttpServletRequest request,HttpServletResponse response)  
throws ServletException, IOException { }
```

Note, this method, as always, takes two parameters: HttpServletRequest and HttpServletResponse and raises two exceptions: ServletException, and IOException.

2)The first thing we always do within the doGet() method is to define the content type for the output which is usually HTTP but could be other formats for example XML or even binary if the servlet is used to generate images. In our example we are outputting HTML and so we also instantiate the out object to which we will be writing our HTML.

```
response.setContentType("text/html");  
PrintWriter out = response.getWriter();
```

Next we want to analyse the query string and pick up the first and last names

```
String sFirstName = request.getParameter("firstname");  
String sLastName = request.getParameter("lastname");
```

And then it is just a matter of writing the appropriate HTML to the output stream

```
out.println("<html>");out.println("<head>");  
out.println("<title>A Simple Servlet</title>");  
out.println("</head>");  
out.println("<body>");out.println("<h1>A Simple Servlet</h1>");  
out.println("<p>Hello " + sFirstName + " " + sLastName + "</p>");  
out.println("</body>");  
out.println("</html>");
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Finally we close the output stream, out.close();

1) **Interface Servlet Class**

Defines methods that all servlets must implement.

A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

To implement this interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.

This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:

1. The servlet is constructed, then initialized with the init method.
2. Any calls from clients to the service method are handled.
3. The servlet is taken out of service, then destroyed with the destroy method, then garbage collected and finalized.

destroy()

Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.

getServletConfig()

Returns a ServletConfig object, which contains initialization and startup parameters for this servlet

getServletInfo()

Returns information about the servlet, such as author, version, and copyright.

init(ServletConfig config)

Called by the servlet container to indicate to a servlet that the servlet is being placed into service.

service(ServletRequest req, ServletResponse res)

Called by the servlet container to allow the servlet to respond to a request.

2) **GenericServlet Class**

java.lang.Object

└─ **javax.servlet.GenericServlet**

destroy()

Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.

getInitParameter(java.lang.String name)

Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.

getServletConfig()

Returns this servlet's ServletConfig object.

getInitParameterNames()

Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.

getServletContext()

Returns a reference to the ServletContext in which this servlet is running.

getServletInfo()

Returns information about the servlet, such as author, version, and copyright.

getServletName()

Returns the name of this servlet instance.

init()

A convenience method which can be overridden so that there's no need to call super.init(config).

service(ServletRequest req, ServletResponse res)

Called by the servlet container to allow the servlet to respond to a request.

log(java.lang.String msg)

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Writes the specified message to a servlet log file, prepended by the servlet's name.

3) **HttpServlet Class**

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. 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
- delete, for HTTP DELETE requests
- init and destroy, to manage resources that are held for the life of the servlet
- getServletInfo, which the servlet uses to provide information about itself

doDelete([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Called by the server (via the service method) to allow a servlet to handle a DELETE request.

doGet([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Called by the server (via the service method) to allow a servlet to handle a GET request

doHead([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Receives an HTTP HEAD request from the protected service method and handles the request.

doOptions([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.

doPost([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Called by the server (via the service method) to allow a servlet to handle a POST request.

doPut([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Called by the server (via the service method) to allow a servlet to handle a PUT request.

doTrace([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Called by the server (via the service method) to allow a servlet to handle a TRACE request.

service([HttpServletRequest](#) req,[HttpServletResponse](#) resp)

Receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class.

ServletException

[ServletException](#) is a general exception that the [servlet](#) container will catch and log. The cause can be anything you define. The exception may contain a root cause exception. [UnavailableException](#) is derived from [ServletException](#) and has the specific meaning that the servlet is not presently available. This may be temporary or permanent. The servlet container should send a 503 error code as a response. Your servlet might throw a [UnavailableException](#) if a database was temporarily off-line.

This exception indicates that a servlet is unavailable. Servlets may report this exception at any time, and the network service running the servlet should behave appropriately. There are two types of unavailability, and sophisticated services will deal with these differently:

Permanent unavailability. The servlet will not be able to handle client requests until some administrative action is taken to correct a servlet problem. For example, the servlet might be misconfigured, or the state of the servlet may be corrupted. Well written servlets will log both the error and the corrective action which an administrator must perform to let the servlet become available.

Temporary unavailability. The servlet can not handle requests at this moment due to a system-wide problem. For example, a third tier server might not be accessible, or there may be insufficient memory or disk storage to handle requests. The problem may be self correcting, such as those due to excessive load, or corrective action may need to be taken by an administrator.

Advance Java Programming

Network services may safely treat both types of exceptions as "permanent", but good treatment of temporary unavailability leads to more robust network services. Specifically, requests to the servlet might be blocked (or otherwise deferred) for a servlet-suggested amount of time, rather than being rejected until the service itself restarts

4) ServletConfig Class

getServletName

java.lang.String **getServletName()**

Returns the name of this servlet instance. The name may be provided via server administration, assigned in the web application deployment descriptor, or for an unregistered (and thus unnamed) servlet instance it will be the servlet's class name.

Returns:the name of the servlet instance

getServletContext

ServletContext **getServletContext()**

Returns a reference to the ServletContext in which the caller is executing.

Returns:a ServletContext object, used by the caller to interact with its servlet container

getInitParameter

java.lang.String **getInitParameter**(java.lang.String name)

Gets the value of the initialization parameter with the given name.

Parameters:name - the name of the initialization parameter whose value to get

Returns:a String containing the value of the initialization parameter, or null if the initialization parameter does not exist

getInitParameterNames

java.util.Enumeration<java.lang.String> **getInitParameterNames()**

Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.

Returns:an Enumeration of String objects containing the names of the servlet's initialization parameters

5) ServletRequest Interface

getAttribute

java.lang.Object **getAttribute**(java.lang.String name)

Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.

Attributes can be set two ways. The servlet container may set attributes to make available custom information about a request. For example, for requests made using HTTPS, the attribute javax.servlet.request.X509Certificate can be used to retrieve information on the certificate of the client. Attributes can also be set programatically using ServletRequest#setAttribute. This allows information to be embedded into a request before a RequestDispatcher call.

Attribute names should follow the same conventions as package names. This specification reserves names matching java.*, javax.*, and sun.*.

Parameters:name - a String specifying the name of the attribute

Returns:an Object containing the value of the attribute, or null if the attribute does not exist

getParameterNames

java.util.Enumeration<java.lang.String> **getParameterNames()**

Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.

Returns:an Enumeration of String objects, each String containing the name of a request parameter; or an empty Enumeration if the request has no parameters

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

getParameterValues

`java.lang.String[] getParameterValues(java.lang.String name)`

Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist. If the parameter has a single value, the array has a length of 1.

Parameters: name - a String containing the name of the parameter whose value is requested

Returns: an array of String objects containing the parameter's values

getRemoteHost

`java.lang.String getRemoteHost()`

Returns the fully qualified name of the client or the last proxy that sent the request. If the engine cannot or chooses not to resolve the hostname (to improve performance), this method returns the dotted-string form of the IP address. For HTTP servlets, same as the value of the CGI variable REMOTE_HOST.

Returns: a String containing the fully qualified name of the client

getRemoteAddr

`java.lang.String getRemoteAddr()`

Returns the Internet Protocol (IP) address of the client or last proxy that sent the request. For HTTP servlets, same as the value of the CGI variable REMOTE_ADDR.

Returns: a String containing the IP address of the client that sent the request

[**getHeaders**](#)(java.lang.String name)

Returns all the values of the specified request header as an Enumeration of String objects.

getQueryString

`java.lang.String getQueryString()`

Returns the query string that is contained in the request URL after the path. This method returns null if the URL does not have a query string. Same as the value of the CGI variable QUERY_STRING.

Returns: a String containing the query string or null if the URL contains no query string. The value is not decoded by the container.

getSession

`HttpSession getSession(boolean create)`

Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

If create is false and the request has no valid HttpSession, this method returns null.

To make sure the session is properly maintained, you must call this method before the response is committed. If the container is using cookies to maintain session integrity and is asked to create a new session when the response is committed, an `IllegalStateException` is thrown.

Parameters: create - true to create a new session for this request if necessary; false to return null if there's no current session

Returns: the HttpSession associated with this request or null if create is false and the request has no valid session

6)HttpServletResponse Class extends [ServletResponse](#) Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response. For example, it has methods to access HTTP headers and cookies.

The servlet container creates an HttpServletResponse object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).

sendRedirect

`void sendRedirect(java.lang.String location)` throws `java.io.IOException`

Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer. The buffer will be replaced with the data set by this method. Calling this method sets the status code to SC_FOUND 302 (Found). This method can accept relative URLs; the servlet container

Advance Java Programming

must convert the relative URL to an absolute URL before sending the response to the client. If the location is relative without a leading '/' the container interprets it as relative to the current request URL. If the location is relative with a leading '/' the container interprets it as relative to the servlet container root.

If the response has already been committed, this method throws an `IllegalStateException`. After using this method, the response should be considered to be committed and should not be written to.

Parameters:location - the redirect location URL

Throws: `java.io.IOException` - If an input or output exception occurs `IllegalStateException` - If the response was committed or if a partial URL is given and cannot be converted into a valid URL

setHeader

void **setHeader**(java.lang.String name, java.lang.String value)

Sets a response header with the given name and value. If the header had already been set, the new value overwrites the previous one. The `containsHeader` method can be used to test for the presence of a header before setting its value.

Parameters:name - the name of the header value - the header value If it contains octet string, it should be encoded according to RFC 2047 (<http://www.ietf.org/rfc/rfc2047.txt>)

setStatus

void **setStatus**(int sc) Sets the status code for this response.

This method is used to set the return status code when there is no error (for example, for the `SC_OK` or `SC_MOVED_TEMPORARILY` status codes).

If this method is used to set an error code, then the container's error page mechanism will not be triggered. If there is an error and the caller wishes to invoke an error page defined in the web application, then `sendError(int, java.lang.String)` must be used instead. This method preserves any cookies and other response headers.

Valid status codes are those in the 2XX, 3XX, 4XX, and 5XX ranges. Other status codes are treated as container specific.

Parameters:sc - the status code

setContentType

void **setContentType**(java.lang.String type)

Sets the content type of the response being sent to the client, if the response has not been committed yet. The given content type may include a character encoding specification, for example, `text/html; charset=UTF-8`. The response's character encoding is only set from the given content type if this method is called before `getWriter` is called.

This method may be called repeatedly to change content type and character encoding. This method has no effect if called after the response has been committed. It does not set the response's character encoding if it is called after `getWriter` has been called or after the response has been committed.

Containers must communicate the content type and the character encoding used for the servlet response's writer to the client if the protocol provides a way for doing so. In the case of HTTP, the `Content-Type` header is used.

Parameters:type - a String specifying the MIME type of the content Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.

encodeURL

java.lang.String **encodeURL**(java.lang.String url)

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged. The implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary.

Advance Java Programming

For robust session tracking, all URLs emitted by a servlet should be run through this method. Otherwise, URL rewriting cannot be used with browsers which do not support cookies.

Parameters:url - the url to be encoded.

Returns:the encoded URL if encoding is needed; the unchanged URL otherwise.

getContentType

java.lang.String **getContentType()**

Returns the content type used for the MIME body sent in this response. The content type proper must have been specified using setContentType(java.lang.String) before the response is committed. If no content type has been specified, this method returns null. If a content type has been specified, and a character encoding has been explicitly or implicitly specified as described in getCharacterEncoding() or getWriter() has been called, the charset parameter is included in the string returned. If no character encoding has been specified, the charset parameter is omitted.

Returns:a String specifying the content type, for example, text/html; charset=UTF-8, or null

getWriter

java.io.PrintWriter **getWriter()** throws java.io.IOException

Returns a PrintWriter object that can send character text to the client. The PrintWriter uses the character encoding returned by getCharacterEncoding(). If the response's character encoding has not been specified as described in getCharacterEncoding (i.e., the method just returns the default value ISO-8859-1), getWriter updates it to ISO-8859-1.

Calling flush() on the PrintWriter commits the response.

Either this method or getOutputStream() may be called to write the body, not both.

Returns:a PrintWriter object that can return character data to the client

Throws:UnsupportedEncodingException - if the character encoding returned by getCharacterEncoding cannot be used
IllegalStateException - if the getOutputStream method has already been called for this response object

java.io.IOException - if an input or output exception occurred

7) SingleThreadModel Class

Ensures that servlets handle only one request at a time. This interface has no methods.

If a servlet implements this interface, you are *guaranteed* that no two threads will execute concurrently in the servlet's service method. The servlet container can make this guarantee by synchronizing access to a single instance of the servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet.

Note that SingleThreadModel does not solve all thread safety issues. For example, session attributes and static variables can still be accessed by multiple requests on multiple threads at the same time, even when SingleThreadModel servlets are used. It is recommended that a developer take other means to resolve those issues instead of implementing this interface, such as avoiding the usage of an instance variable or synchronizing the block of the code accessing those resources. This interface is deprecated in Servlet API version 2.4.

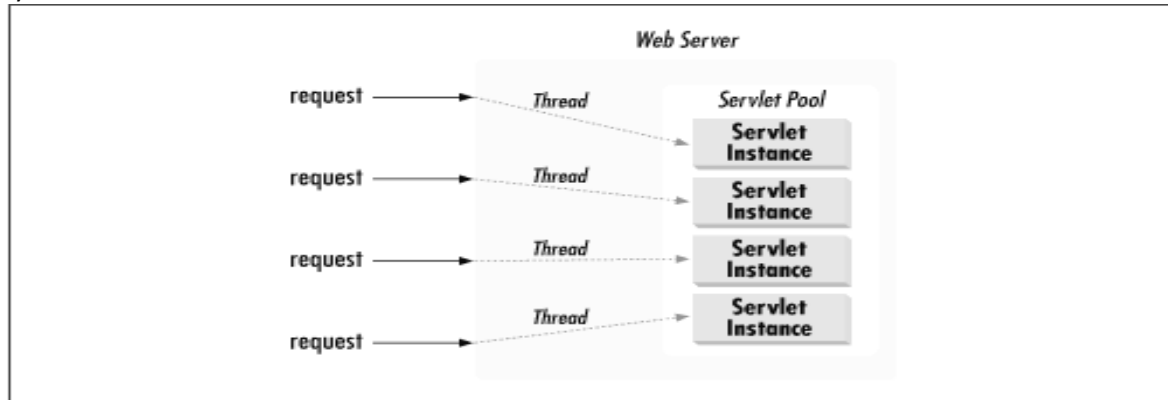
Single-Thread Model

Although it is standard to have one servlet instance per registered servlet name, it is possible for a servlet to elect instead to have a pool of instances created for each of its names, all sharing the duty of handling requests. Such servlets indicate this desire by implementing the javax.servlet.SingleThreadModel interface. This is an empty, tag interface that defines no methods or variables and serves only to flag the servlet as wanting the alternate life cycle.

A server that loads a SingleThreadModel servlet must guarantee, according to the Servlet API documentation, "that no two threads will execute concurrently the service method of that servlet." To accomplish this, each thread uses a free servlet instance from the pool, as shown in Figure. Thus, any

Advance Java Programming

servlet implementing SingleThreadModel can be considered thread safe and isn't required to synchronize access to its instance variables.



Such a life cycle is pointless for a counter or other servlet application that requires central state maintenance. The life cycle can be useful, however, in avoiding synchronization while still performing efficient request handling.

For example, a servlet that connects to a database sometimes needs to perform several database commands atomically as part of a single transaction. Normally, this would require the servlet to synchronize around the database commands (letting it manage just one request at a time) or to manage a pool of database connections where it can "check out" and "check in" connections (letting it support multiple concurrent requests). By instead implementing SingleThreadModel and having one "connection" instance variable per servlet, a servlet can easily handle concurrent requests by letting its server manage the servlet instance pool (which doubles as a connection pool). The skeleton code is shown in [Example](#).

Example . Handling database connections using SingleThreadModel

```
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SingleThreadConnection extends HttpServlet
    implements SingleThreadModel {
    Connection con = null; // database connection, one per pooled servlet instance
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        // Establish the connection for this instance
        con = establishConnection();
        con.setAutoCommit(false);
    }
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        // Use the connection uniquely assigned to this instance
        Statement stmt = con.createStatement();
        // Update the database any number of ways
        // Commit the transaction
    }
}
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
        con.commit();
    }
    public void destroy() {
        if (con != null) con.close();
    }
    private Connection establishConnection() {
        // Not implemented. See Chapter 9, "Database Connectivity".
    }
}
```

Session Tracking with servlet API

1) public interface **HttpSession**

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to

- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
- Bind objects to sessions, allowing user information to persist across multiple user connections

When an application stores an object in or removes an object from a session, the session checks whether the object implements `HttpSessionBindingListener`. If it does, the servlet notifies the object that it has been bound to or unbound from the session. Notifications are sent after the binding methods complete. For session that are invalidated or expire, notifications are sent after the session has been invalidated or expired.

When container migrates a session between VMs in a distributed container setting, all session attributes implementing the `HttpSessionActivationListener` interface are notified.

A servlet should be able to handle cases in which the client does not choose to join a session, such as when cookies are intentionally turned off. Until the client joins the session, `isNew` returns true. If the client chooses not to join the session, `getSession` will return a different session on each request, and `isNew` will always return true.

Session information is scoped only to the current web application (`ServletContext`), so information stored in one context will not be directly visible in another.

getAttribute

java.lang.Object **getAttribute**(java.lang.String name)

Returns the object bound with the specified name in this session, or null if no object is bound under the name.

Parameters: name - a string specifying the name of the object

Returns: the object with the specified name

Throws: `IllegalStateException` - if this method is called on an invalidated session

getAttributeNames

java.util.Enumeration<java.lang.String> **getAttributeNames**()

Returns an Enumeration of String objects containing the names of all the objects bound to this session.

Returns: an Enumeration of String objects specifying the names of all the objects bound to this session

Throws: `IllegalStateException` - if this method is called on an invalidated session

getId

java.lang.String **getId**()

Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.

Returns: a string specifying the identifier assigned to this session

getLastAccessedTime

long **getLastAccessedTime()**

Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.

Actions that your application takes, such as getting or setting a value associated with the session, do not affect the access time.

Returns: a long representing the last time the client sent a request associated with this session, expressed in milliseconds since 1/1/1970 GMT

Throws: `IllegalStateException` - if this method is called on an invalidated session

getCreationTime

long **getCreationTime()**

Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

Returns: a long specifying when this session was created, expressed in milliseconds since 1/1/1970 GMT

Throws: `IllegalStateException` - if this method is called on an invalidated session

isNew

boolean **isNew()**

Returns true if the client does not yet know about the session or if the client chooses not to join the session. For example, if the server used only cookie-based sessions, and the client had disabled the use of cookies, then a session would be new on each request.

Returns: true if the server has created a session, but the client has not yet joined

Throws: `IllegalStateException` - if this method is called on an already invalidated session

removeAttribute

void **removeAttribute**(java.lang.String name)

Removes the object bound with the specified name from this session. If the session does not have an object bound with the specified name, this method does nothing.

After this method executes, and if the object implements `HttpSessionBindingListener`, the container calls `HttpSessionBindingListener.valueUnbound`. The container then notifies any `HttpSessionAttributeListeners` in the web application.

Parameters: name - the name of the object to remove from this session

Throws: `IllegalStateException` - if this method is called on an invalidated session

setAttribute

void **setAttribute**(java.lang.String name, java.lang.Object value)

Binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced.

After this method executes, and if the new object implements `HttpSessionBindingListener`, the container calls `HttpSessionBindingListener.valueBound`. The container then notifies any `HttpSessionAttributeListeners` in the web application.

If an object was already bound to this session of this name that implements `HttpSessionBindingListener`, its `HttpSessionBindingListener.valueUnbound` method is called.

If the value passed in is null, this has the same effect as calling `removeAttribute()`.

Parameters: name - the name to which the object is bound; cannot be null value - the object to be bound

Throws: `IllegalStateException` - if this method is called on an invalidated session

setMaxInactiveInterval

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

void setMaxInactiveInterval(int interval)

Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

An interval value of zero or less indicates that the session should never timeout.

Parameters:interval - An integer specifying the number of seconds

invalidate

void invalidate()

Invalidates this session then unbinds any objects bound to it.

Throws: IllegalStateException - if this method is called on an already invalidated session

2) public interface ServletContext

Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

There is one context per "web application" per Java Virtual Machine. (A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace such as /catalog and possibly installed via a .war file.)

In the case of a web application marked "distributed" in its deployment descriptor, there will be one context instance for each virtual machine. In this situation, the context cannot be used as a location to share global information (because the information won't be truly global). Use an external resource like a database instead.

The ServletContext object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.

getContext

ServletContext **getContext**(java.lang.String uripath)

Returns a ServletContext object that corresponds to a specified URL on the server.

This method allows servlets to gain access to the context for various parts of the server, and as needed obtain RequestDispatcher objects from the context. The given path must be begin with /, is interpreted relative to the server's document root and is matched against the context roots of other web applications hosted on this container.

In a security conscious environment, the servlet container may return null for a given URL.

Parameters:uripath - a String specifying the context path of another web application in the container.

Returns:the ServletContext object that corresponds to the named URL, or null if either none exists or the container wishes to restrict this access.

getRequestDispatcher

RequestDispatcher **getRequestDispatcher**(java.lang.String path)

Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path. A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

The pathname must begin with a / and is interpreted as relative to the current context root. Use getContext to obtain a RequestDispatcher for resources in foreign contexts.

This method returns null if the ServletContext cannot return a RequestDispatcher.

Parameters:path - a String specifying the pathname to the resource

Returns:a RequestDispatcher object that acts as a wrapper for the resource at the specified path, or null if the ServletContext cannot return a RequestDispatcher

setAttribute

void setAttribute(java.lang.String name,java.lang.Object object)

Binds an object to a given attribute name in this ServletContext. If the name specified is already used for an attribute, this method will replace the attribute with the new to the new attribute.

If listeners are configured on the ServletContext the container notifies them accordingly.

Advance Java Programming

If a null value is passed, the effect is the same as calling `removeAttribute()`.

Attribute names should follow the same convention as package names. The Java Servlet API specification reserves names matching `java.*`, `javax.*`, and `sun.*`.

Parameters:name - a String specifying the name of the attribute object - an Object representing the attribute to be bound

getAttribute

`java.lang.Object getAttribute(java.lang.String name)`

Returns the servlet container attribute with the given name, or null if there is no attribute by that name.

An attribute allows a servlet container to give the servlet additional information not already provided by this interface. See your server documentation for information about its attributes. A list of supported attributes can be retrieved using `getAttributeNames`.

The attribute is returned as a `java.lang.Object` or some subclass.

Attribute names should follow the same convention as package names. The Java Servlet API specification reserves names matching `java.*`, `javax.*`, and `sun.*`.

Parameters:name - a String specifying the name of the attribute

Returns:an Object containing the value of the attribute, or null if no attribute exists matching the given name

getInitParameter

`java.lang.String getInitParameter(java.lang.String name)`

Returns a String containing the value of the named context-wide initialization parameter, or null if the parameter does not exist.

This method can make available configuration information useful to an entire web application. For example, it can provide a webmaster's email address or the name of a system that holds critical data.

Parameters:name - a String containing the name of the parameter whose value is requested

Returns:a String containing at least the servlet container name and version number

getInitParameterNames

`java.util.Enumeration<java.lang.String> getInitParameterNames()`

Returns the names of the context's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the context has no initialization parameters.

Returns:an Enumeration of String objects containing the names of the context's initialization parameters

getServerInfo

`java.lang.String getServerInfo()`

Returns the name and version of the servlet container on which the servlet is running.

The form of the returned string is *servername/versionnumber*. For example, the JavaServer Web Development Kit may return the string JavaServer Web Dev Kit/1.0.

The servlet container may return other optional information after the primary string in parentheses, for example, JavaServer Web Dev Kit/1.0 (JDK 1.1.6; Windows NT 4.0 x86).

Returns:a String containing at least the servlet container name and version number

removeAttribute

`void removeAttribute(java.lang.String name)`

Removes the attribute with the given name from this ServletContext. After removal, subsequent calls to `getAttribute(java.lang.String)` to retrieve the attribute's value will return null.

If listeners are configured on the ServletContext the container notifies them accordingly.

Parameters:name - a String specifying the name of the attribute to be removed

Session Tracking

Very few web applications are confined to a single page, so having a mechanism for tracking users through a site can often simplify application development. The Web, however, is an inherently stateless environment. A client makes a request, the server fulfills it, and both promptly forget about

Advance Java Programming

each other. In the past, applications that needed to deal with a user through multiple pages (for instance, a shopping cart) had to resort to complicated dodges to hold onto state information, such as hidden fields in forms, setting and reading cookies, or rewriting URLs to contain state information.

1) URL Rewriting :

You can use another way of session tracking that is URL rewriting where your browser does not support cookies. It is mechanism by which the requested URL is modified to include extra information. This extra information can be in the form of extra path info, added parameters or some other URL change.

Since for URL rewriting, provided space is limited so the extra information is limited to a unique session ID.

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class UrlRewritingExample extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        java.io.PrintWriter out = response.getWriter();
        String contextPath = request.getContextPath();
        String encodedUrl = response
            .encodeURL(contextPath + "/WelcomePage.jsp");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>URL Rewriter</title>");
        out.println("</head>");
        out.println("<body><center>");
        out.println("<h2>URL rewriting Example</h2>");
        out.println("For welcome page - <a href=\"\" + encodedUrl
            + "\"> Click Here</a>.");
        out.println("</center></body>");
        out.println("</html>");
    }
}
```

2) Hidden Field :

Session tracking is a technique to maintain about the requests of a same user for a specific time period. You can maintain session tracking in three ways - Hidden form field, Cookies, URL rewriting.

Hidden form field is a way of session tracking so that you can save the information in client browser itself. For that you can use hidden field for securing fields. So at the time of page display, no one can see these fields. When client submit the form, the browser also transfer these hidden value to the server with other fields.

You can secure your field as –

<input type =?hidden? name =?name? value=?value?/>

In servlet you can get hidden form field as -

String name= request.getParameter(?name?);

Advance Java Programming

Example : In this example, we are mentioning two form field hidden which are name and location in **HiddenFieldPage.html** and accessing these field in to our servlet **HiddenFieldServlet.java**.

HiddenFieldPage.html -

```
<html>
<head>
<title>Hidden Field Example</title>
</head>
<body>
<center>
<h2>Click on Submit to check hidden field </h2>
<form action="HiddenFieldServlet" method="post">
<input type="hidden" name="name" value="Roseindia">
<input type="hidden" name="location" value="New Delhi">
<input type="submit" value="Submit"></form>
</center></body></html>
```

HiddenFieldServlet.java –

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HiddenFieldServlet extends HttpServlet {
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>");out.println("<body>");out.println("<center>");
String name = request.getParameter("name");
String location = request.getParameter("location");
out.println("<form action=HiddenFieldPage.jsp>");
out.println("<b>Company Name : </b>" + name);
out.println("<br><b>Locaion : </b>" + location);
out.println("<br><br><b>Click on GoBack to go on JSP page </b>");
out.println("<br><input type=submit value='GoBack'>");
out.println("</form>");out.println("</center>");out.println("</body>");
}
}
```

3) Cookies

You can use HTTP cookies to store information about a shopping session, and each subsequent connection can look up the current session and then extract information about that session from some location on the server machine. For example, a servlet could do something like the following:

```
String sessionID = makeUniqueString();
Hashtable sessionInfo = new Hashtable();
Hashtable globalTable = findTableStoringSessions();
globalTable.put(sessionID, sessionInfo);
Cookie sessionCookie = new Cookie("JSESSIONID", sessionID);
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
sessionCookie.setPath("/");  
response.addCookie(sessionCookie);
```

Then, in later requests the server could use the globalTable hash table to associate a session ID from the JSESSIONID cookie with the sessionInfo hash table of data associated with that particular session. This is an excellent solution and is the most widely used approach for session handling. Still, it is nice that servlets have a higher-level API that handles all this plus the following tedious tasks:

Extracting the cookie that stores the session identifier from the other cookies (there may be many cookies, after all).

Setting an appropriate expiration time for the cookie.

Associating the hash tables with each request.

Generating the unique session identifiers

Session tracking with servlet API with HttpSession Interface

public Object getAttribute(String name)

The third session method returns an array of the current bound names stored in the session. This method is convenient if you want to remove all the current bindings in a session. Its signature is listed as follows:

public String[] getAttributeNames()

The last session method is the removeAttribute() method. As its name suggests, it removes a binding from the current session. It takes a string parameter representing the name associated with the binding. Its method signature is listed as follows:

public void removeAttribute(String name)

public void setMaxInactiveInterval(int interval)

This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

public void setAttribute(String name, Object value)

This method binds an object to this session, using the name specified.

public boolean isNew()

This method returns true if the client does not yet know about the session or if the client chooses not to join the session

public long getLastAccessedTime()

This method returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

public String getId()

This method returns a string containing the unique identifier assigned to this session.

public long getCreationTime()

This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

public Enumeration getAttributeNames()

This method returns an Enumeration of String objects containing the names of all the objects bound to this session

public void invalidate()

This method invalidates this session and unbinds any objects bound to it.

Servlet Collaboration

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

Advance Java Programming

There are two ways to forward control from one page to another.

1.Using RequestDispatcher

Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server.

The servlet container creates the RequestDispatcher object, which is used as a wrapper around a server resource located at a particular path or given by a particular name.

Method Summary:

void forward(ServletRequest request, ServletResponse response)

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

void include(ServletRequest request, ServletResponse response)

Includes the content of a resource (servlet, JSP page, HTML file) in the response.

In next example Dispatcher1 servlet forward request to Dispatcher2 servlet—

Dispatcher1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Dispatcher1 extends HttpServlet {
    public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        RequestDispatcher rd=req.getRequestDispatcher("Dispatcher2");
        rd.include(req,res);
    }
}
```

Dispatcher2.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Dispatcher2 extends HttpServlet {
    public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("I am output of Dispatcher2.java");
    }
}
```

Output:

I am output of Dispatcher2.java

2.Using sendRedirect()

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RedirectDemo1 extends HttpServlet {
    public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException, IOException {
        res.sendRedirect("http://localhost:7001/servletdemo/");
    }
}
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
RedirectDemo2");  
}  
}
```

Servlet Context

ServletContext is a interface which helps us to communicate with the servlet container. There is only one ServletContext for the entire web application and the components of the web application can share it. The information in the ServletContext will be common to all the components. Remember that each servlet will have its own **ServletConfig**. The ServletContext is created by the container when the web application is deployed and after that only the context is available to each servlet in the web application.

Web application initialization:

1. First of all the web container reads the deployment descriptor file and then creates a name/value pair for each <context-param> tag.
2. After creating the name/value pair it creates a new instance of ServletContext.
3. Its the responsibility of the Container to give the reference of the ServletContext to the context init parameters.
4. The servlet and jsp which are part of the same web application can have the access of the ServletContext.

The Context init parameters are available to the entire web application not just to the single servlet like servlet init parameters.

getContext

ServletContext **getContext**(java.lang.String uripath)

Returns a ServletContext object that corresponds to a specified URL on the server.

This method allows servlets to gain access to the context for various parts of the server, and as needed obtain RequestDispatcher objects from the context. The given path must be begin with /, is interpreted relative to the server's document root and is matched against the context roots of other web applications hosted on this container.

In a security conscious environment, the servlet container may return null for a given URL.

Parameters: uripath - a String specifying the context path of another web application in the container.

Returns: the ServletContext object that corresponds to the named URL, or null if either none exists or the container wishes to restrict this access.

getRequestDispatcher

RequestDispatcher **getRequestDispatcher**(java.lang.String path)

Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path. A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

The pathname must begin with a / and is interpreted as relative to the current context root. Use getContext to obtain a RequestDispatcher for resources in foreign contexts.

This method returns null if the ServletContext cannot return a RequestDispatcher.

Parameters: path - a String specifying the pathname to the resource

Returns: a RequestDispatcher object that acts as a wrapper for the resource at the specified path, or null if the ServletContext cannot return a RequestDispatcher

getServerInfo

java.lang.String **getServerInfo**()

Returns the name and version of the servlet container on which the servlet is running.

The form of the returned string is *servername/versionnumber*. For example, the JavaServer Web Development Kit may return the string JavaServer Web Dev Kit/1.0.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

The servlet container may return other optional information after the primary string in parentheses, for example, JavaServer Web Dev Kit/1.0 (JDK 1.1.6; Windows NT 4.0 x86).

Returns: a String containing at least the servlet container name and version number

getInitParameter

java.lang.String **getInitParameter**(java.lang.String name)

Returns a String containing the value of the named context-wide initialization parameter, or null if the parameter does not exist.

This method can make available configuration information useful to an entire web application. For example, it can provide a webmaster's email address or the name of a system that holds critical data.

Parameters: name - a String containing the name of the parameter whose value is requested

Returns: a String containing at least the servlet container name and version number

getInitParameterNames

java.util.Enumeration<java.lang.String> **getInitParameterNames**()

Returns the names of the context's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the context has no initialization parameters.

Returns: an Enumeration of String objects containing the names of the context's initialization parameters

getAttribute

java.lang.Object **getAttribute**(java.lang.String name)

Returns the servlet container attribute with the given name, or null if there is no attribute by that name.

An attribute allows a servlet container to give the servlet additional information not already provided by this interface. See your server documentation for information about its attributes. A list of supported attributes can be retrieved using `getAttributeNames`.

The attribute is returned as a `java.lang.Object` or some subclass.

Attribute names should follow the same convention as package names. The Java Servlet API specification reserves names matching `java.*`, `javax.*`, and `sun.*`.

Parameters: name - a String specifying the name of the attribute

Returns: an Object containing the value of the attribute, or null if no attribute exists matching the given name

setAttribute

void **setAttribute**(java.lang.String name, java.lang.Object object)

Binds an object to a given attribute name in this ServletContext. If the name specified is already used for an attribute, this method will replace the attribute with the new to the new attribute.

If listeners are configured on the ServletContext the container notifies them accordingly.

If a null value is passed, the effect is the same as calling `removeAttribute()`.

Attribute names should follow the same convention as package names. The Java Servlet API specification reserves names matching `java.*`, `javax.*`, and `sun.*`.

Parameters: name - a String specifying the name of the attribute

object - an Object representing the attribute to be bound

removeAttribute

void **removeAttribute**(java.lang.String name)

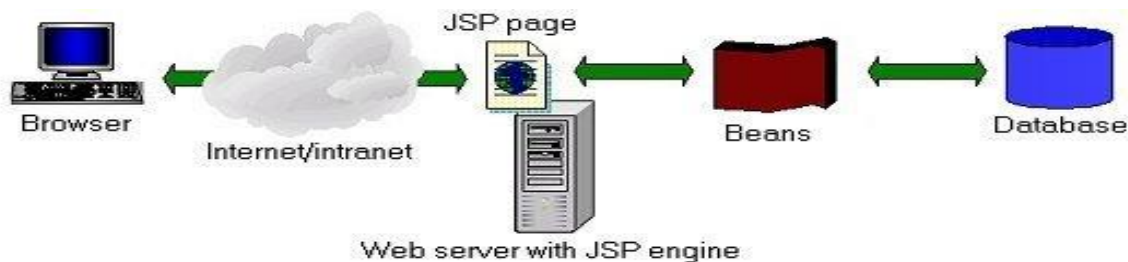
Removes the attribute with the given name from this ServletContext. After removal, subsequent calls to `getAttribute(java.lang.String)` to retrieve the attribute's value will return null.

If listeners are configured on the ServletContext the container notifies them accordingly.

Parameters: name - a String specifying the name of the attribute to be removed

When a request is mapped to a JSP page, the web container first checks whether the JSP page's servlet is older than the JSP page. If the servlet is older, the web container translates the JSP page into a servlet class and compiles the class. During development, one of the advantages of JSP pages over servlets is that the build process is performed automatically.

- Goal: Create dynamic web content (HTML, XML, ...) for a Web Application
- Goal: Make it easier/cleaner to mix static HTML parts with dynamic Java servlet code
- JSP specification ver. 2.0
- JSP is a presentation layer technology that sits on top of a Java servlets model and makes working with HTML easier. Like SSJS, it allows you to mix static HTML content with server-side scripting to produce dynamic output. By default, JSP uses Java as its scripting language; however, the specification allows other languages to be used, just as ASP can use other languages (such as JavaScript and VBScript). While JSP with Java will be more flexible and robust than scripting platforms based on simpler languages like JavaScript and VBScript, Java also has a steeper learning curve than simple scripting languages.
- To offer the best of both worlds - a robust web application platform and a simple, easy-to-use language and tool set - JSP provides a number of server-side tags that allow developers to perform most dynamic content operations without ever writing a single line of Java code. So developers who are only familiar with scripting, or even those who are simply HTML designers, can use JSP tags for generating simple output without having to learn Java. Advanced scripters or Java developers can also use the tags, or they can use the full Java language if they want to perform advanced operations in JSP pages.



JavaServer Pages, or JSPs, are a simple but powerful technology used most often to generate dynamic HTML on the server side. It is used to simplify the creation and management of the dynamic web pages by providing a more convenient authoring framework than servlet. JSPs are a direct extension of Java servlets designed to let the developer embed Java logic directly into a requested document. JSP pages combines static markup like HTML and XML, with special scripting tags. JSP pages resembles markup documents, but each JSP page is translated into a servlet the first time it is invoked. A JSP document must end with the extension.jsp.

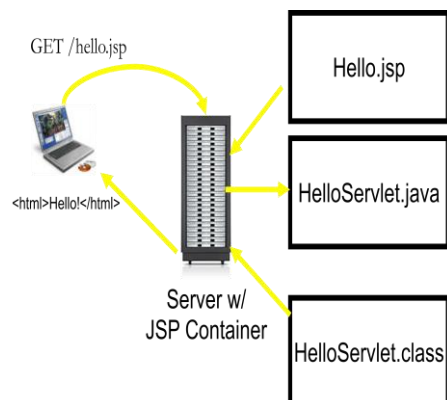
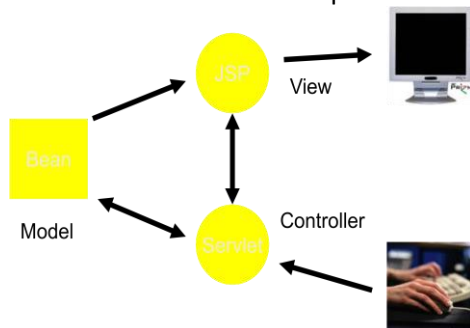
JSP BASICS

- CGI & Servlets -- Mostly Code with some HTML via print & out.println
- JSP/ASP/PHP -- Mostly HTML, with code snippets thrown in
 - No explicit recompile
 - Great for small problems
 - Easier to program
 - Not for large computations
- Mostly HTML page, with extension .jsp
- Include JSP tags to enable dynamic content creation

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

- Translation: JSP → Servlet class
- Compiled at Request time
(first request, a little slow)
- Execution: Request → JSP Servlet's service method
- `<html>`
- `<body>`
- `<jsp:useBean.../>`
- `<jsp:getProperty.../>`
- `<jsp:getProperty.../>`
- `</body>`
- `</html>`
- Code -- Computation
- HTML -- Presentation
- Separation of Roles
 - Developers
 - Content Authors/Graphic Designers/Web Masters
 - Supposed to be cheaper... but not really...
- A Design Pattern
- Controller -- receives user interface input, updates data model
- Model -- represents state of the world (e.g. shopping cart)
- View -- looks at model and generates an appropriate user interface to present the data and allow for further input



JSP pages designed and developed less like programs and more like web pages. JSP pages are ideal when we need to display markup with embedded dynamic content. However, although generating HTML is much easier than with a servlet, JSP pages are less suited to handling processing logic. JSP pages can use JavaBeans to achieve a clean separation of static content and the java code that produce dynamic web applications.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

JSP and ASP :

JSP and ASP are fairly similar in the functionality that they provide. JSP may have slightly higher learning curve. Both allow embedded code in an HTML page, session variables and database access and manipulation. Whereas ASP is mostly found on Microsoft platforms i.e. NT, JSP can operate on any platform that conforms to the J2EE specification. JSP allow component reuse by using Javabeans and EJBs. ASP provides the use of COM / ActiveX controls.

JSP compared to ASP.NET :

ASP.NET is based on the Microsoft .NET framework. The .NET framework allows applications to be developed using different programming languages such as Visual Basic, C# and JavaScript. JSP and Java still has the advantage that it is supported on many different platforms and the Java community has many years of experience in designing and developing Enterprise quality scalable applications. This is not to say that ASP.NET is bad, actually it is quite an improvement over the old asp code.

JSP compared to Servlets:

A Servlet is a Java class that provides special server side service. It is hard work to write HTML code in Servlets. In Servlets you need to have lots of println statements to generate HTML. JSP pages are converted to Servlets so actually can do the same thing as old Java Servlets.

This allows JSP pages to be created and maintained by designers with presentation skills, they do not need to know java.

While there is an overlap between their capabilities, think of servlets as controller objects, and JSP pages as view objects. Don't think that we need to make a choice between using servlets and JSP pages in a web application; they are complementary technologies, and a complex web application will use both.

A simple JSP page that display current date in an HTML page would like this:

```
<%@ page import="java.util.Date" %>
<html>
<body>
The current time is <%= (new Date()).toString() %>
</body>
</html>
```

Save this file as **DateDemo.jsp** in root directory like Webdemo and make request like <http://localhost:8085/Webdemo/DateDemo.jsp>

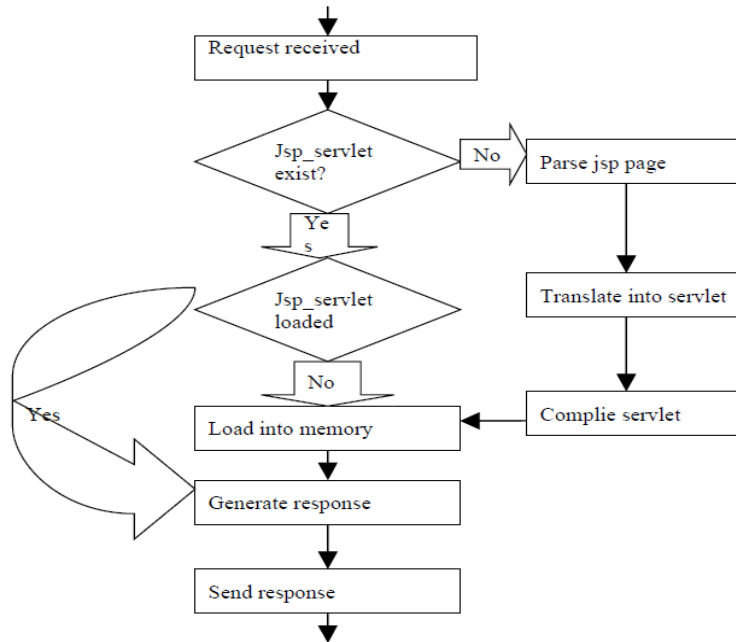
The first time a JSP is loaded by the JSP container, the servlet code necessary to implement the JSP tags is automatically generated, compiled, and loaded into the servlet container.

This occurs at translation time. It is important to note that this occurs only the first time a JSP page is requested. There will be a slow response the first time a JSP page is accessed, but on subsequent requests the previously compiled servlet simply processes the requests. This occurs at run time.

If we modify the source code for the JSP, it is automatically recompiled and reloaded the next time that page is requested.

This figure summaries the process from request to response:

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming



JSP LIFE CYCLE

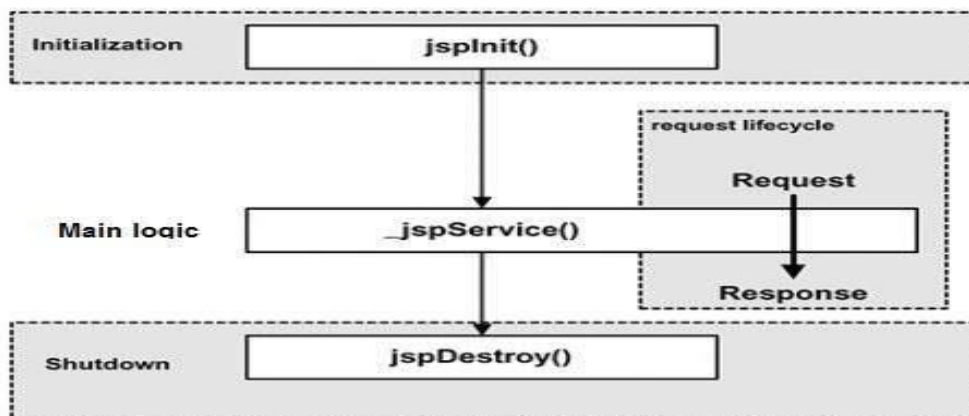
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP –

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



JSP Compilation

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method –

```
public void jspInit(){  
  
    // Initialization code...  
  
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The **_jspService()** method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows –

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {  
  
    // Service handling code...  
  
}
```

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The **jspDestroy()** method has the following form –

```
public void jspDestroy() {  
  
    // Your cleanup code goes here.  
  
}
```

}

JSP Elements:

- 1. Directives Elements:** These affects the overall structure of the servlet that results from translation, but produce no output themselves.
- 2. Scripting Elements:** JSP scripting elements let we insert code into the servlet that will be generated from the JSP page.
- 3. Action Elements:** These are special tags available to affect the runtime behavior of the JSP page. JSP standard actions are predefined custom tags that can be used to encapsulate common actions easily. There are two types of JSP standard actions: the first type is related to JavaBean functionality, and the second type consists of all other standard actions.

Directives Elements

JSP directives are JSP elements that provide global information about a JSP page. An example would be a directive that included a list of Java classes to be imported into a JSP.

Three possible directives are currently defined by the JSP specification page, include, and taglib.

1)Page Directive

The page directive defines information that will globally affect the JSP containing the directive. The syntax of a JSP page directive is

`<%@ page attribute="value" %>`

Note Because all mandatory attributes are defaulted, we are not required to specify any page directives.

Attributes for the page Directive are:

language="scriptingLanguage"

Tells the server which language will be used to compile the JSP file. Java is currently the only available JSP language, but we hope there will be other language support in the not-toodistant future.

extends="className"

Defines the parent class from which the JSP will extend. While we can extend JSP from other servlets, doing so will limit the optimizations performed by the JSP/servlet engine and is therefore not recommended.

import="importList"

Defines the list of Java packages that will be imported into this JSP. It will be a commaseparated list of package names and fully qualified Java classes.

session="true|false"

Determines whether the session data will be available to this page. The default is true. If our JSP is not planning on using the session, then this attribute should be set to false for better performance.

buffer="none|size in kb"

Determines whether the output stream is buffered. The default value is 8KB.

autoFlush="true|false"

Determines whether the output buffer will be flushed automatically, or whether it will throw an exception when the buffer is full. The default is true.

isThreadSafe="true|false"

Tells the JSP engine that this page can service multiple requests at one time. By default, this value is true. If this attribute is set to false, the SingleThreadModel is used.

info="text"

Advance Java Programming

Represents information about the JSP page that can be accessed by invoking the page's *Servlet.getServletInfo()* method.

errorPage="error_url" Represents the relative URL to a JSP that will handle JSP exceptions.
isErrorPage="true|false"

States whether the JSP is an errorPage. The default is false.

contentType="ctinfo"

Represents the MIME type and character set of the response sent to the client.

Example

Next listing presents a page that uses a class not in the standard JSP import list: *java.util.Date*

```
<%@ page import="java.util.Date" %>
<html><body>
The current time is <%= (new Date()).toString() %>
</body></html>
```

Output:

Current date

Next listing presents a page that uses *contentType* attribute to generating excel spreadsheets.

```
<%@ page contentType="application/vnd.ms-excel" %>
<!-- Note that there are tabs, not spaces, between columns. --%>
1997 1998 1999 2000 2001
12.3 13.4 14.5 15.6 16.7
```

Use of error Page and isErrorPage attribute:-

Error1.jsp : generate an arithmetic exception

```
<%@ page language="java" ErrorPage="Error2.jsp" %>
<html><body><center>
<%out.println(0/0);%>
</center></body></html>
```

Error2.jsp

```
<%@ page language="java" isErrorPage="true" %>
<html><body>
<center>This is a error page<hr><br><br>
<%=exception%>
</center>
</body>
</html>
```

Output:



2) Include Directive

The include directive is used to insert text and/or code at JSP translation time. The syntax of the include directive is shown in the following code snippet:

```
<%@ include file="relativeURLspec" %>
```

The file attribute can reference a normal text HTML file or a JSP file, which will be evaluated at translation time. This resource referenced by the file attribute must be local to the Web application that

Advance Java Programming

contains the include directive. Here's a sample include directive:
`<%@ include file="header.jsp" %>`

Note: Because the include directive is evaluated at translation time, this included text will be evaluated only once. Thus, if the included resource changes, these changes will not be reflected until the JSP/servlet container is restarted or the modification date of the JSP that includes that file is changed.

3) taglib Directive

The taglib directive states that the including page uses a custom tag library, uniquely identified by a URI and associated with a prefix that will distinguish each set of custom tags to be used in the page.

JSP introduced an extremely valuable new capability, the ability to define our own JSP tags. We define how the tag, its attributes, and its body are interpreted. In order to use custom JSP tags, we need to define three separate components: the tag handler class that defines the tag's behavior, the tag library descriptor file that maps the XML element names to the tag implementations, and the JSP file that uses the tag library.

Directive Element

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html> <head> <title>JSP is Easy</title> </head>
<body bgcolor="white">
<h1>JSP is as easy as ...</h1> <!-- Calculate the sum of 1 + 2 + 3 dynamically --%>
1 + 2 + 3 = <c:out value="${1 + 2 + 3}" /> </body></html>
```

Page directive

```
<%@ page import="package.class" %>
<%@ page import="java.util.*" %>
<%@ page contentType="text/html" %>
<% response.setContentType("text/html"); %>
```

Include directive

```
<%@ include file="Relative URL">
```

Included at Translation time

May contain JSP code such as response header settings, field definitions, etc... that affect the main page

Scripting Elements

1)Declarations

JSP declarations are used to define Java variables and methods in a JSP. A JSP declaration must be a complete declarative statement. JSP declarations are initialized when the JSP page is first loaded. After the declarations have been initialized, they are available to other declarations, expressions, and scriptlets within the same JSP.

The syntax for a JSP declaration is as follows:

```
<%! declaration %>
```

A sample variable declaration using this syntax is shown here:

```
<%! String name = new String("BOB"); %>
```

A sample method declaration using the same syntax is as follows:

```
<%! public String getName() { return name; } %>
```

To get a better understanding of declarations, let's take the previous string declaration and embed it into a JSP document. The sample document would look similar to the following code snippet:

```
<HTML><BODY>
<%! String name = new String("BOB"); %>
</BODY></HTML>
```


Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

When this document is initially loaded, the JSP code is converted to servlet code and the name declaration is placed in the declaration section of the generated servlet. It is now available to all other JSP components in the JSP. Note It should be noted that all JSP declarations are defined at the class level, in the servlet generated from the JSP, and will therefore be evaluated prior to all JSP expressions and scriptlet code.

Expressions

JSP expressions are JSP components whose text, upon evaluation by the container, is replaced with the resulting value of the container evaluation. JSP expressions are evaluated at request time, and the result is inserted at the expression's referenced position in the JSP file. If the resulting expression cannot be converted to a string, then a translation-time error will occur. If the conversion to a string cannot be detected during translation, a `ClassCastException` will be thrown at request time.

The syntax of a JSP expression is as follows:

```
<%= expression %>
```

A code snippet containing a JSP expression is shown here:

```
Hello <B><%= getName() %></B>
```

Here is a sample JSP document containing a JSP expression:

```
<HTML><BODY>
```

```
<%! public String getName() { return "Bob"; } %>
```

```
Hello <B><%= getName() %></B>
```

```
</BODY></HTML>
```

Scriptlets

Scriptlets are the JSP components that bring all the JSP elements together. They can contain almost any coding statements that are valid for the language referenced in the language directive.

They are executed at request time, and they can make use of all the JSP components. The syntax for a scriptlet is as follows:

```
<% scriptlet source %>
```

When JSP scriptlet code is converted into servlet code, it is placed into the generated servlet's `service()` method. The following code snippet contains a simple JSP that uses a scripting element to print the text "Hello Bob" to the requesting client:

```
<HTML><BODY>
```

```
<% out.println("Hello Bob"); %>
```

```
</BODY></HTML>
```

We should note that while JSP scriptlet code can be very powerful, composing all our JSP logic using scriptlet code can make our application difficult to manage. This problem led to the creation of custom tag libraries.

Action Elements

Script Element

```
<%= expr %>
```

```
<%! decl %>
```

```
<% code %>
```

Action Element

Standard Actions

JSTL (tag library) Actions

Custom Actions

Custom actions: created with tag libraries

1) forward Syntax : `<jsp:forward page="relativeURLspec" />`

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Allows the runtime dispatch of the request to one of the following in the same context as the current page:

static resource (html) (for GET requests), A JSP, A Java servlet.

Terminates the execution of the current page

Example : `<% String dest = "/location/" + someValue; %>`

`<jsp:forward page='<%= dest %>' />`

2) include Syntax: `<jsp:include page="urlSpec" flush="true|false"/>`
and

`<jsp:include page="urlSpec" flush="true|false">`

`<jsp:param /> }* </jsp:include>`

Include a resource at **request** time, or a request with parameters

3) param Syntax: `<jsp:param name="name" value="value" />`

Used with include and forward and params elements

4) plugin

replaces the tag with either an `<object>` or `<embed>` tag as needed by the user agent to display an applet or similar plugin properly. The params and fallback actions are used to provide for passing parameters to the plugin and providing feedback to the user if the plugin failed to load.

Example:

```
<jsp:plugin type=applet code="Molecule.class" codebase="/html" >
  <jsp:params><jsp:param name="molecule" value="molecules/benzene.mol"/>
</jsp:params><jsp:fallback>
  <p> unable to start plugin </p>
</jsp:fallback></jsp:plugin>
```

Jsp Comment

We can use two types of comments with the JSP page one is for HTML and another for JSP. Standard HTML comments have this form: `<!--this comment will appear in the client's browser -->` and **JSP – specific comments that use this syntax: `< %-- this comment will not appear in the client browser --%>`** JSP comments will not appear in the page output to the client.

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri=http://java.sun.com/jstl/core %>
<html> <head> <title>JSP is Easy</title> </head>
<body bgcolor="white">
<h1>JSP is as easy as ...</h1>
<%-- Calculate the sum of 1 + 2 + 3 dynamically --%>
1 + 2 + 3 = <c:out value="\${1 + 2 + 3}" />
</body></html>
```

Jsp template:

In JSP page, everything that is not a directive, declaration, scriptlets, expression, action element, or JSP comment is termed template data. Usually all the HTML and text in the page. In other words, template data is ignored by the JSP translator. The data is output to the client as if it had appeared within a static web page.

```
<html><head> <title>JSP is Easy</title></head>
<body bgcolor="white">
<h1>JSP is as easy as ...</h1>
<%-- Calculate the sum of 1 + 2 + 3 dynamically --%>
1 + 2 + 3 = <c:out value="\${1 + 2 + 3}" />
</body></html>
```

SCOPE OF THE JSP OBJECTS

Objects that are created as part of a jsp page have a certain lifetime and may or may not be accessible to other components or objects in the web application. The lifetime and accessibility of an object is known as scope. In some cases, such as with the implicit objects discussed in the next section, the scope is set and cannot be changed. With other objects, you can set the scope of the object. There are four valid scopes :

- Page Scope
- Request Scope
- Session Scope
- Application Scope

1. Page Scope:

This scope is the most restrictive. With page scope, the object is accessible only within the current jsp page in which it is defined. JavaBeans created with page scope and objects created by scriptlets are thread-safe. JSP implicit objects out, exception, response, config, pageContext and page have 'page' scope.

2. Request Scope:

JSP object created using the 'request' scope can be accessed from any pages that serve that request. This means that the object is available within the page in which it is created, and within pages to which the request is forwarded or included. Objects with request scope are thread-safe. Only the execution thread for a particular request can access these objects. The JSP object will be bound to the request object. Implicit object request has the 'request' scope.

3. Session Scope:

Objects with session scope are available to all application components that participate in the client's session. These objects are not thread-safe. If multiple requests could use the same session object at the same time, you must synchronize access to that object. The jsp object that is created using the session scope is bound to the session object. Implicit object session has the 'session' scope.

4. Application Scope:

This is the least restrictive scope. Objects that are created with application scope are available to the entire application for the life of the application. These objects are not thread-safe, and access to them must be synchronized if there is a chance that multiple requests will attempt to change the object at the same time. The JSP object is bound to the application object. Implicit object application has the 'application' scope.

Using of Implicit Object

Objects exist in different scopes:

- page scope: a single page
- request scope: pages processing the same request
- session scope: pages processing requests that are in the same session
- application scope: all pages in an application

Remember, an object created inside a method would only have scope in that method as well. In Java, we also limit scope by using the { }

An object with a given scope is accessible to pages that are in the same scope as the page where the object was created. (more on this later)

Certain objects are created implicitly for each page by the container:

All of these are predefined and can be used inside your JSP.

- request
- response
- pageContext
- session
- application

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

- out
- config
- page
- exception - for error pages only

request

- A reference to the protocol dependent subclass of javax.servlet.ServletRequest (i.e. javax.servlet.http.HttpServletRequest)
- You can use it just like you did with servlets
- request scope

response

- A reference to the protocol dependent subclass of javax.servlet.ServletResponse (i.e. javax.servlet.http.HttpServletResponse)
- You can use it just like you did with servlets
- page scope

pageContext

- an object that provides a context to store references to objects used by the page
- encapsulates implementation-dependent features, and provides convenience methods.:
getOut(), getException(), getPage() getRequest(), getResponse(), getSession(),
getServletConfig() and getServletContext().
- provides uniform access to diverse objects
setAttribute(), getAttribute(), findAttribute(), removeAttribute(), getAttributesScope()
and getAttributeNamesInScope().
- provides methods for inclusion, error handling, and forwarding
forward(), include(), and handlePageException().
- A mechanism for obtaining the JspWriter for output
- A single object to manage the various scoped namespaces
- page scope

session

- A reference to the HTTP session (javax.servlet.http.HttpSession)
- You can use it just like you did with servlets.
- Only available if this page is marked as participating in a session. (How we do this is later). If it isn't participating, referring to the session is illegal and results in a translation error
- session scope

application

- The servlet context returned by calling getServletConfig().getContext() in a servlet
- You can use it as you would the context in a servlet
- application scope

out

- javax.servlet.jsp.JspWriter
- An object that writes into the output stream like the PrintWriter in a servlet (it is a subclass).
- page scope

config

- The javax.servlet.ServletConfig that you would get from calling getServletConfig() in a servlet
- use for getting init parameters
- page scope

page

- a java.lang.Object
- The instance of this page' implementation class processing the current request

Advance Java Programming

- When using Java as the scripting language, then page is a synonym for this
- page scope
- not typically used

exception

- a java.lang.Throwable
- only available on error pages (we'll talk about how to create these next)
- The exception that resulted in the error page being invoked

Special tags that affect the out stream and use, modify, or create objects

Handling Errors and Exception

- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation

Page directive

The errorPage Attribute:

The errorPage attribute tells the JSP engine which page to display if there is an error while the current page runs. The value of the errorPage attribute is a relative URL.

The following directive displays MyErrorPage.jsp when all uncaught exceptions are thrown:

```
<%@ page errorPage="MyErrorPage.jsp" %>
```

The isErrorPage Attribute:

The isErrorPage attribute indicates that the current JSP can be used as the error page for another JSP.

The value of isErrorPage is either true or false. The default value of the isErrorPage attribute is false.

For example, the handleError.jsp sets the isErrorPage option to true because it is supposed to handle errors:

```
<%@ page isErrorPage="true" %>
```

JSP gives you an option to specify Error Page for each JSP. Whenever the page throws an exception, the JSP container automatically invokes the error page.

Following is an example to specify an error page for a main.jsp. To set up an error page, use the `<%@ page errorPage="xxx" %>` directive.

```
<%@ page errorPage="ShowError.jsp" %>
<html><head> <title>Error Handling Example</title></head>
<body>
<%
    // Throw an exception to invoke the error page
    int x = 1;
    if (x == 1)
    {
        throw new RuntimeException("Error condition!!!");
    }
%></body></html>
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

JavaBeans are reusable software components for Java. They are classes that encapsulate many objects into a single object (the bean). They are serializable, have a 0-argument constructor, and allow access to properties using getter and setter methods.

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBeans from other Java classes:

- It provides a default, no-argument constructor.
- It should be serializable and implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

JavaBeans Properties:

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including classes that you define.

A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class:

Method	Description
getPropertyName()	For example, if property name is <i>firstName</i> , your method name would be getFirstName() to read that property. This method is called accessor.
setPropertyName()	For example, if property name is <i>firstName</i> , your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method

Advantages

- The properties, events, and methods of a bean that are exposed to another application can be controlled.
- A bean may register to receive events from other objects and can generate events that are sent to those other objects.
- Auxiliary software can be provided to help configure a java bean.
- The configuration setting of bean can be saved in a persistent storage and restored at a later time.

Disadvantages

- A class with a nullary constructor is subject to being instantiated in an invalid state. If such a class is instantiated manually by a developer (rather than automatically by some kind of framework), the developer might not realize that the class has been improperly instantiated. The compiler can't detect such a problem, and even if it's documented, there's no guarantee that the developer will see the documentation.
- Having to create a getter for every property and a setter for many, most, or all of them can lead to an immense quantity of boilerplate code.

JavaBeans API

The JavaBeans functionality is provided by a set of classes and interfaces in the java.beans package.

Interface	Description
-----------	-------------

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

Interface	Description
AppletInitializer	Methods in this interface are used to initialize Beans that are also applets.
BeanInfos	This interface allows the designer to specify information about the events, methods and properties of a Bean.
Customizer	This interface allows the designer to provide a graphical user interface through which a bean may be configured.
DesignMode	Methods in this interface determine if a bean is executing in design mode.
ExceptionListener	A method in this interface is invoked when an exception has occurred.
PropertyChangeListener	A method in this interface is invoked when a bound property is changed.
PropertyEditor	Objects that implement this interface allow the designer to change and display property values.
VetoableChangeListener	A method in this interface is invoked when a Constrained property is changed.
Visibility	Methods in this interface allow a bean to execute in environments where the GUI is not available.

The use of the MVC architecture is to separate the business logic and application data from the presentation data to the user. Here are the reasons why we should use the MVC design pattern.

1. They are reusable: When the problem recurs, there is no need to invent a new solution, we just have to follow the pattern and adapt it as necessary.

2. They are expressive: By using the MVC design pattern our application becomes more expressive.

1) Model:

The model object knows about all the data that needs to be displayed. It is the model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not aware about the presentation data and how that data will be displayed to the browser.

2) View:

The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains the same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the view's to maintain the consistency in its presentation when the model changes.

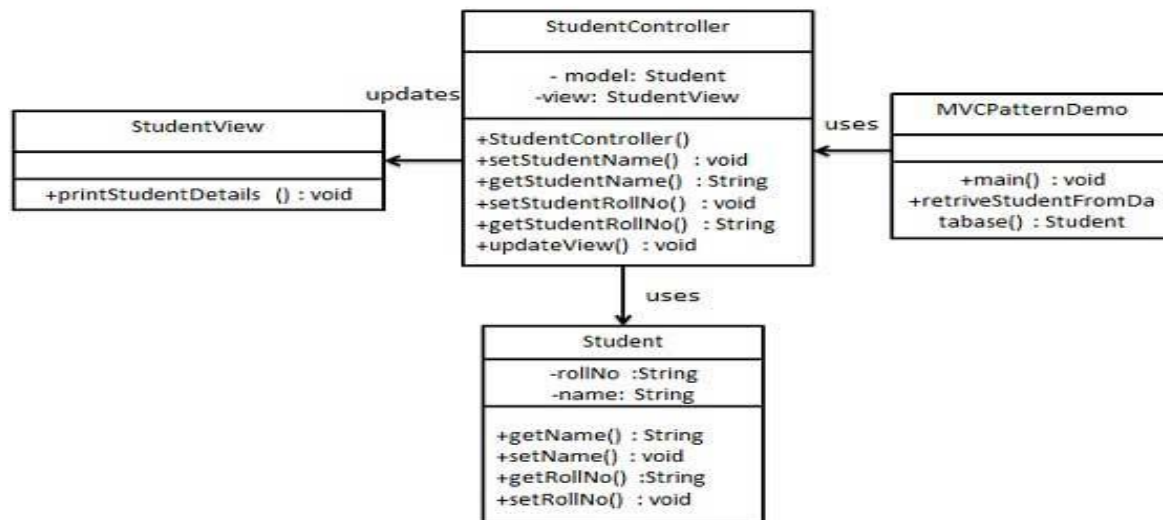
3) Controller:

Whenever the user sends a request for something then it always goes through the controller. The controller is responsible for intercepting the requests from the view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUIs, the views and the controllers often work very closely together.

Implementation

We're going to create a *Student* object acting as a model. *StudentView* will be a view class which can print student details on console and *StudentController* is the controller class responsible to store data in *Student* object and update view *StudentView* accordingly.

MVCPatternDemo, our demo class will use *StudentController* to demonstrate use of MVC pattern.



Step 1

Create Model.

Student.java

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
public class Student {  
    private String rollNo;  
    private String name;  
    public String getRollNo() {  
        return rollNo;  
    }  
    public void setRollNo(String rollNo) {  
        this.rollNo = rollNo;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Step 2

Create View.

StudentView.java

```
public class StudentView {  
    public void printStudentDetails(String studentName, String studentRollNo){  
        System.out.println("Student: ");  
        System.out.println("Name: " + studentName);  
        System.out.println("Roll No: " + studentRollNo);  
    }  
}
```

Step 3

Create Controller.

StudentController.java

```
public class StudentController {  
    private Student model;  
    private StudentView view;  
  
    public StudentController(Student model, StudentView view){  
        this.model = model;  
        this.view = view;  
    }  
    public void setStudentName(String name){  
        model.setName(name);  
    }  
    public String getStudentName(){  
        return model.getName();  
    }  
  
    public void setStudentRollNo(String rollNo){  
        model.setRollNo(rollNo);  
    }  
    public String getStudentRollNo(){
```

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
        return model.getRollNo();
    }
    public void updateView(){
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```

Step 4

Use the *StudentController* methods to demonstrate MVC design pattern usage.

MVCPatternDemo.java

```
public class MVCPatternDemo {
    public static void main(String[] args) {

        //fetch student record based on his roll no from the database
        Student model = retrieveStudentFromDatabase();

        //Create a view : to write student details on console
        StudentView view = new StudentView();

        StudentController controller = new StudentController(model, view);
        controller.updateView();
        //update model data
        controller.setStudentName("John");
        controller.updateView();
    }
    private static Student retrieveStudentFromDatabase(){
        Student student = new Student();
        student.setName("Robert");
        student.setRollNo("10");
        return student;
    }
}
```

Step 5

Verify the output.

Student:

Name: Robert

Roll No: 10

Student:

Name: John

Roll No: 10

EJB

The EJB 3.0 specification makes it easier than ever to develop Enterprise JavaBeans, perhaps encouraging you to consider developing EJBs for the first time. If that is the case, congratulations, you have successfully avoided the many pitfalls of EJB developers before you, and can enjoy the ease of EJB 3.0 development. But before you start development, you might want to know what Enterprise

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

JavaBeans are and what purpose they serve. This article explains the basics of EJBs and how you can utilize them in a J2EE application.

What is an EJB?

An Enterprise JavaBeans (EJB) is a reusable, portable J2EE component. EJBs consist of methods that encapsulate business logic. For example, an EJB may have business logic that contains a method to update customer data in a database. A variety of remote and local clients can invoke this method. Additionally, EJBs run inside a container, allowing the developer to focus on the business logic contained in the bean without having to worry about complicated and error-prone issues such as transaction support, security, and remote object access. EJBs are developed as *POJOs*, or Plain Old Java Objects, and developers can use metadata annotations to specify to the container how these beans are to be managed.

What are the advantages of EJB model?

- The development of EJB applications is easy as the business logic is separated by the application developer and at the same time the developer can utilize the services of the EJB container.
- EJB are components, where the EJB vendors vend these components which encapsulate the functionality according to the need. Application development made easy as the EJB specification ensures the usability of beans developed by others can also be used in a specific application.
- The isolation of labor in developing, deploying, administering, providers made it faster to develop an EJB application.
- The major operations of managing transactions, state, multithreading, connection pooling etc. will be managed by the EJB container. The security is also provided by the EJB container.
- The EJB architecture is compatible with other APIs like servlets and JSPs.

What are the limitations of EJB models?

- The no file I/O is a security concern. By loading files using `Class.getResource()` and bundling non-EJB config files with EJB-jar.
- While mapping a CMP bean with SQL Server database, a field of float type needs to be mapped with database column of type FLOAT only, not for REAL.
- A table which has a multi-column primary key and one or more foreign keys which share any one of the primary key columns, then the foreign keys must contain columns which are not of primary key or exactly the same primary key columns.

For e.g., the two columns that comprises a primary key are A and B. Then the columns that comprise a foreign key are column A and column B or column C and column D, but certainly not column A and column D or column B and column C or column A and column C or column B and D.

- While using root-leaf approach in order to map inheritance, the foreign-key constraints are to be removed from the database for avoiding referential integrity related issues. The same process is to be followed while using secondary maps with multiple tables.

Types of EJBs

EJBs consist of three main types: Session, Entity, and Message-Driven. A Session bean performs a distinct, de-coupled task, such as checking credit history for a customer. An Entity bean is a complex business entity which represents a business object that exists in the database. A Message-Driven bean is used to receive asynchronous JMS messages. Let's examine these types further:

Session Beans

Session beans generally represent actions in the business process such as "process order." Session beans are classified based on the maintenance of the conversation state, either stateful or stateless.

Stateless session beans do not have an internal state. They do not keep track of the information that is passed from one method call to another. Thus, each invocation of a stateless business method is independent of its previous invocation; for example, calculating taxes or shipping charges. When a method to calculate tax charges is invoked with a certain taxable value, the tax value is calculated and returned to the calling method, without the necessity to store the caller's internal state for future invocation. Because they do not maintain state, these beans are simple to manage for the container. When the client requests a stateless bean instance, it may receive an instance from the pool of stateless session bean instances that are maintained by the container. Also, because stateless session beans can be shared, the container can maintain a lesser number of instances to serve a large number of clients. To specify that a Java Bean is to be deployed and managed as a stateless session bean, simply add the annotation `@Stateless` to the bean.

A stateful session bean maintains a conversational state across method invocations; for example, an online shopping cart of a customer. When the customer starts online shopping, the customer's details are retrieved from the database. The same details are available for the other methods that are invoked when the customer adds or removes items from the cart, places the order, and so on. Yet, stateful session beans are transient because the state does not survive session termination, system crash, or network failure. When a client requests a stateful session bean instance, that client is assigned one stateful instance, and the state of the bean is maintained for that client. To specify to the container that a stateful session bean instance should be removed upon the completion of a certain method, add the annotation `@Remove` to the method.

Session Bean Example

```
import javax.ejb.Stateless.*;
/**
 * A simple stateless session bean implementing the incrementValue() method of the * CalculateEJB
 interface.
 */
@Stateless(name="CalculateEJB")
public class CalculateEJBBean
implements CalculateEJB
{
int value = 0;
public String incrementValue()
{
value++;
return "value incremented by 1";
}
}
```

Entity Beans

An entity bean is an object that manages persistent data, potentially uses several dependent Java objects, and can be uniquely identified by a primary key. Specify that a class is an entity bean by including the `@Entity` annotation. Entity beans represent persistent data from the database, such as a row in a customer table, or an employee record in an employee table. Entity beans are also sharable across multiple clients. For example, an employee entity bean can be used by various clients to calculate the annual salary of an employee or to update the employee address. Specific fields of the entity bean object can be made persistent. All fields in the entity bean not marked with the `@Transient` annotation

Advance Java Programming

are considered persistent. A key feature of EJB 3.0 is the ability to create Entity beans which contain object/relational mappings using metadata annotations. For example, to specify that an Entity bean's empId field is mapped to the EMPNO attribute of the Employees table, annotate the table name using @Table(name="Employees") and the field using @Column(name="EMPNO"), as shown in the example below. Additionally, a special feature of EJB 3.0 is that you can easily test Entity beans during development, as running Entity beans outside a container is now possible using the Oracle Application Server Entity Test Harness.

Entity Bean Example

```
import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;
@Entity
@Table(name = "EMPLOYEES")
public class Employee implements java.io.Serializable
{
    private int empId;
    private String eName;
    private double sal;
    @Id
    @Column(name="EMPNO", primaryKey=true)
    public int getEmpId()
    {
        return empId;
    }
    public void setEmpId(int empId)
    {
        this.empId = empId;
    }
    public String getEname()
    {
        return eName;
    }
    public void setName(String eName)
    {
        this.eName = eName;
    }

    public double getSal()
    {
        return sal;
    }

    public void setSal(double sal)
    {
        this.sal = sal;
    }
    public String toString()
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
{
StringBuffer buf = new StringBuffer();
buf.append("Class:")
.append(this.getClass().getName()).append(" :: ").append(" empld:").append(getEmpld()).append("
ename:").append(getEname()).append("sal:").append(getSal());
return buf.toString();
}
}
```

Message-Driven Beans

Message-driven beans (MDBs) provide an easier method to implement asynchronous communication than by using straight Java Message Services (JMS). MDBs were created to receive asynchronous JMS messages. The container handles most of the setup processes that are required for JMS queues and topics. It sends all the messages to the interested MDB. An MDB allows J2EE applications to send asynchronous messages that can then be processed by the application. To specify that a bean is an MDB, implement the `javax.jms.MessageListener` interface and annotate the bean with `@MessageDriven`.

Message Driven Bean Example

```
import javax.ejb.MessageDriven;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.Inject;
import javax.jms.*;
import java.util.*;
import javax.ejb.TimerObject;
import javax.ejb.Timer;
import javax.ejb.TimerService;
@MessageDriven(
activationConfig = {
@ActivationConfigProperty(propertyName="connectionFactoryJndiName",
propertyValue="jms/TopicConnectionFactory"),
@ActivationConfigProperty(propertyName="destinationName", propertyValue="jms/myTopic"),
@ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Topic"),
@ActivationConfigProperty(propertyName="messageSelector", propertyValue="RECIPIENT = 'MDB'")
}
)
/**
 * A simple Message-Driven Bean that listens to the configured JMS Queue or Topic and gets notified via
 an * invocation of it's onMessage() method when a message has been posted to the Queue or Topic.
 The bean
 * prints the contents of the message.
 */
public class MessageLogger implements MessageListener, TimerObject
{
@Inject javax.ejb.MessageDrivenContext mc;
public void onMessage(Message message)
{
```

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
System.out.println("onMessage() - " + message);
try
{
String subject = message.getStringProperty("subject");
String inmessage = message.getStringProperty("message");
System.out.println("Message received\n\tDate: " + new java.util.Date() + "\n\tSubject: " + subject +
"\n\tMessage: " + inmessage + "\n");
System.out.println("Creating Timer a single event timer");
TimerService ts = mc.getTimerService();
Timer timer = ts.createTimer(30000, subject);
System.out.println("Timer created by MDB at: " + new Date(System.currentTimeMillis()) + " with info:
"+subject);
}
catch (Throwable ex)
{
ex.printStackTrace();
}
}
public void ejbTimeout(Timer timer)
{
System.out.println("EJB 3.0: Timer with MDB");
System.out.println("ejbTimeout() called at: " + new Date(System.currentTimeMillis()));
return;
}
}
```

Timer Service:

Applications that model business work flows often rely on timed notifications. The timer service of the enterprise bean container enables you to schedule timed notifications for all types of enterprise beans except for stateful session beans. You can schedule a timed notification to occur according to a calendar schedule, at a specific time, after a duration of time, or at timed intervals. For example, you could set timers to go off at 10:30 a.m. on May 23, in 30 days, or every 12 hours.

Enterprise bean timers are either programmatic timers or automatic timers. **Programmatic timers** are set by explicitly calling one of the timer creation methods of the TimerService interface. **Automatic timers** are created upon the successful deployment of an enterprise bean that contains a method annotated with the java.ejb.Schedule or java.ejb.Schedules annotations.

Hibernate

Introduction to Hibernate:

Hibernate is popular open source object relational mapping tool for Java platform. It provides powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition and the Java collections framework. The *Hibernate Query Language*, designed as a "minimal" object-oriented extension to SQL, provides an elegant bridge between the object and relational worlds. Hibernate also allows you to express queries using native SQL or Java-based Criteria and Example queries. Hibernate is now the most popular object/relational mapping solution for Java.

Need for hibernate

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

ORM stands for Object/Relational Mapping. Hibernate is open source ORM tool. ORM is technology to access the data from the database. With the help of ORM framework, business object is directly mapped to the database table. Hibernate reduces the time to perform database operations. It is implemented by using simple POJOs.

ORM tools like hibernate provides following benefits: -

- Improved performance-Lazy loading sophisticated caching, and Eager loading.
- Improved Productivity: High-level object oriented API, Less Java code to write, No SQL to write.
- Improved maintainability: less code to write.
- Improved portability: generates database ?specific SQL.

Features of Hibernate

- Hibernate 3.0 provides three full-featured query facilities: **Hibernate Query Language**, the newly enhanced **Hibernate Criteria Query API**, and enhanced support for queries expressed in the **native SQL** dialect of the database.
- Filters for working with temporal (historical), regional or permissioned data.
- Enhanced Criteria query API: with full support for projection/aggregation and subselects.
- Runtime performance monitoring: via JMX or local Java API, including a second-level cache browser.
- Eclipse support, including a suite of Eclipse plug-ins for working with Hibernate 3.0, including mapping editor, interactive query prototyping, schema reverse engineering tool.
- Hibernate is Free under LGPL: Hibernate can be used to develop/package and distribute the applications for free.
- Hibernate is Scalable: Hibernate is very performant and due to its dual-layer architecture can be used in the clustered environments.
- Less Development Time: Hibernate reduces the development timings as it supports inheritance, polymorphism, composition and the Java Collection framework.
- Automatic Key Generation: Hibernate supports the automatic generation of primary key for your.
- JDK 1.5 Enhancements: The new JDK has been released as a preview earlier this year and we expect a slow migration to the new 1.5 platform throughout 2004. While Hibernate3 still runs perfectly with JDK 1.2, Hibernate3 will make use of some new JDK features. JSR 175 annotations, for example, are a perfect fit for Hibernate metadata and we will embrace them aggressively. We will also support Java generics, which basically boils down to allowing type safe collections.
- EJB3-style persistence operations: EJB3 defines the create() and merge() operations, which are slightly different to Hibernate's saveOrUpdate() and saveOrUpdateCopy() operations. Hibernate3 will support all four operations as methods of the Session interface. Hibernate XML binding enables data to be represented as XML and POJOs interchangeably.
- The EJB3 draft specification support for POJO persistence and annotations.

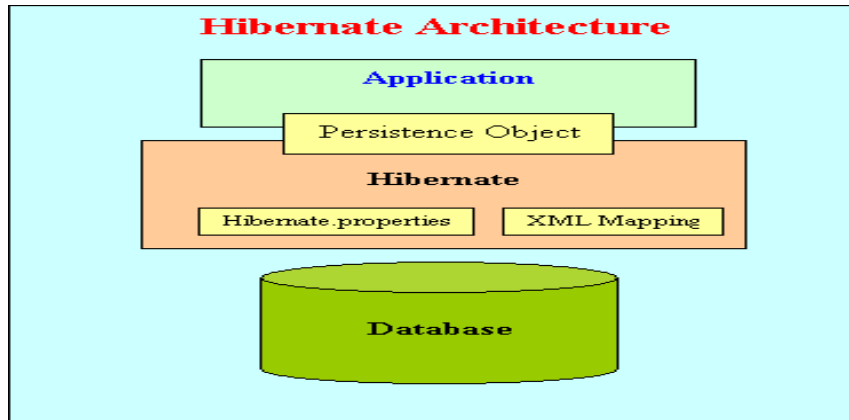
Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Disadvantages of Hibernate:

- ❑ **Lots of API to learn:** A lot of effort is required to learn Hibernate. So, not very easy to learn hibernate easily.
- ❑ **Debugging:** Sometimes debugging and performance tuning becomes difficult.
- ❑ **Slower than JDBC:** Hibernate is slower than pure JDBC as it is generating lots of SQL statements in runtime.
- ❑ **Not suitable for Batch processing:** It advisable to use pure JDBC for batch processing.

Hibernate Architecture



The above diagram shows that Hibernate is using the database and configuration data to provide persistence services (and persistent objects) to the application.

To use Hibernate, it is required to create Java classes that represents the table in the database and then map the instance variable in the class with the columns in the database. Then Hibernate can be used to perform operations on the database like select, insert, update and delete the records in the table. Hibernate automatically creates the query to perform these operations.

Hibernate architecture has three main components:

- **Connection Management**
Hibernate Connection management service provide efficient management of the database connections. Database connection is the most expensive part of interacting with the database as it requires a lot of resources of open and close the database connection.
- **Transaction management:**
Transaction management service provide the ability to the user to execute more than one database statements at a time.
- **Object relational mapping:**
Object relational mapping is technique of mapping the data representation from an object model to a relational data model. This part of the hibernate is used to select, insert, update and delete the records form the underlying table. When we pass an object to a **Session.save()** method, Hibernate reads the state of the variables of that object and executes the necessary query.

Hibernate is very good tool as far as object relational mapping is concern, but in terms of connection management and transaction management, it is lacking in performance and capabilities. So usually hibernate is being used with other connection management and transaction management tools. For example apache DBCP is used for connection pooling with the Hibernate.

Hibernate provides a lot of flexibility in use. It is called "**Lite**" architecture when we only uses the object relational mapping component. While in "**Full Cream**" architecture all the three component Object Relational mapping, Connection Management and Transaction Management) are used

Downloading and Configuring and necessary files to Hibernate in Eclipse

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Downloading the latest version of Hibernate

The latest version of Hibernate can be downloaded from its official website at <http://hibernate.org/orm/downloads/>. At the time of writing of the tutorial the final version of the Hibernate was Hibernate 4.2.8.Final.

Visit the website at <http://hibernate.org/orm/downloads/> and you will find the links to download the software. Here is the screen shot of the same:

Hibernate ORM Downloads

stable 4.3.5.Final

Interested in commercial support? Check out Red Hat's offering.







Releases

- 4.3.5.Final**  2014-04-02 **stable**
Maven gav: org.hibernate:hibernate-core:4.3.5.Final
JPA 2.1 support
[More on this release](#)
- 4.2.13.Final**  2014-05-28 **stable**
Maven gav: org.hibernate:hibernate-core:4.2.13.Final
ORM maintenance release, JPA 2.0
[More on this release](#)

We have downloaded the file **hibernate-release-4.2.8.Final.zip**. It includes the libraries necessary to run the Hibernate based applications.

Installing Hibernate

Copy the downloaded file (**hibernate-release-4.2.8.Final.zip**) into a directory and then unzip it. Here the screen shot of the content of the file:







Name	Date modified	Type	Size
 documentation	6/11/2014 11:30 AM	File folder	
 lib	6/11/2014 11:33 AM	File folder	
 project	6/11/2014 11:33 AM	File folder	
 changelog	4/2/2014 10:06 AM	Text Document	338 KB
 hibernate_logo	1/8/2014 5:12 PM	GIF image	2 KB
 lgpl	1/8/2014 5:12 PM	Text Document	26 KB

Here you will find documentation, lib and project directories. Documentation directory contains the documentation and tutorials. The lib directory contains the library files and project directory contains many example projects.

Here is the content of lib directory:











Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming






Name	Date modified	Type
 envers	6/11/2014 11:33 AM	File folder
 jpa	6/11/2014 11:33 AM	File folder
 jpa-metamodel-generator	6/11/2014 11:33 AM	File folder
 optional	6/11/2014 11:33 AM	File folder
 osgi	6/11/2014 11:33 AM	File folder
 required	6/11/2014 11:33 AM	File folder

The lib directory contains the required folder which contains the required library of Hibernate. It contains other libraries also.

Here is the screen shot of lib/required folder:

Name	Date modified	Type	Size
 antlr-2.7.7	1/8/2014 5:23 PM	Executable Jar File	435 KB
 dom4j-1.6.1	1/8/2014 5:15 PM	Executable Jar File	307 KB
 hibernate-commons-annotations-4.0.4.F...	1/8/2014 5:23 PM	Executable Jar File	74 KB
 hibernate-core-4.3.5.Final	4/2/2014 10:43 AM	Executable Jar File	5,108 KB
 hibernate-jpa-2.1-api-1.0.0.Final	1/8/2014 5:23 PM	Executable Jar File	111 KB
 jandex-1.1.0.Final	1/8/2014 5:23 PM	Executable Jar File	75 KB
 javassist-3.18.1-GA	1/8/2014 5:15 PM	Executable Jar File	698 KB
 jboss-logging-3.1.3.GA	1/8/2014 5:22 PM	Executable Jar File	56 KB
 jboss-logging-annotations-1.2.0.Beta1	1/8/2014 5:22 PM	Executable Jar File	12 KB
 jboss-transaction-api_1.2_spec-1.0.0.Final	1/8/2014 5:23 PM	Executable Jar File	28 KB

Documentation directory contains following sub-directories:

Name	Date modified	Type
 devguide	6/11/2014 11:27 AM	File folder
 javadocs	6/11/2014 11:30 AM	File folder
 manual	6/11/2014 11:30 AM	File folder
 quickstart	6/11/2014 11:30 AM	File folder
 topical	6/11/2014 11:30 AM	File folder


Installing Hibernate is very easy process, you just have to unzip and then copy the library files from its lib directory. If you are using Eclipse, you have to copy the files into lib directory of your project. After copy the file in your project's lib directory, add the lib files in the class build path.

For copying the file into build path:

- Right click on the project
- Select "Java Build Path"
- Then on the right side select "Libraries" and then click on the "Add JARs" and then select the jar files from your project.

This will make the jar files available in your project.

Jars files of hibernate:

 Anttr-2.7.6.jar

asm.jar

asm-attrs.jar

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

cglib-2.1.3.jar
commons-collections-2.1.1.jar
commons-logging-1.0.4.jar
ehcash.jar
dom4j-1.6.1.jar
hibernate3.jar
jta.jar
log4j-1.2.3.jar

These are the main jar files to run hibernate related programming and among all the jars hibernate3.jar is the main file, but for annotation we need to add 4 – 6 other jar files, i will let you when time comes.

Remember: along with the hibernate jars we must include one more jar file, which is nothing but related to our database, this is depending on your database.

So finally we need total of **12 jar files** to run the hibernate related program.

Hibernate Configuration file

The Hibernate framework uses the **hibernate.cfg.xml** and other optional files for configuring the ORM runtime. This article is discussing about the configuration files of Hibernate. As a beginner you should be aware of all these files. You will be dealing with all these file while developing your Hibernate based applications.

Following is the list of configuration files in Hibernate based project:

- **hibernate.cfg.xml:** This file is currently used by the developers for providing ORM related configuration.
- **hibernate.properties:** In the older version of Hibernate this file was used for configuring ORM parameters. But now a days developers are using hibernate.cfg.xml file.
- **<pojo-table-mapping>.xml:** This file is used to map the POJO class with the table and it's field. After introduction of JPA annotations developers are rarely using this file.

Hibernate configuration file hibernate.cfg.xml

The hibernate.cfg.xml is xml file used to provide the variables necessary for configuring the Hibernate ORM runtime. In most of the cases the default value is ok for a general project. The things you will have to change is the dialect, database url, database username and database password.

Here is a simple example of the file **hibernate.cfg.xml**:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">
jdbc:mysql://localhost/hibernatetutorial</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password"></property>
<property name="hibernate.connection.pool_size">10</property>
<property name="show_sql">true</property>
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

Advance Java Programming

```
<property name="hibernate.hbm2ddl.auto">update</property>
<!-- Mapping files -->
<mapping resource="contact.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Here are the important properties of the file:

- hibernate.connection.driver_class - The JDBC Driver class
- hibernate.connection.url - Database URL
- hibernate.connection.username - Database user name
- hibernate.connection.password - Password to connect to database
- hibernate.connection.pool_size - Default connection pool size
- show_sql - If it is true it will show all the sql generated by Hibernate
- dialect - Database dialect

The **hibernate.cfg.xml** should present in the class-path of the project.

Hibernate Mapping file

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables.

Though many Hibernate users choose to write the XML by hand, a number of tools exist to generate the mapping document. These include **XDoclet**, **Middlegen** and **AndroMDA** for advanced Hibernate users.

Let us consider our previously defined POJO class whose objects will persist in the table defined in next section.

```
public class Employee
{
    private int id;
    private String firstName;
    private String lastName;
    private int salary;

    public Employee() {}
    public Employee(String fname, String lname, int salary)
    {
        this.firstName = fname;
        this.lastName = lname;
        this.salary = salary;
    }
    public int getId() {
        return id;
    }
    public void setId( int id ) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName( String first_name ) {
        this.firstName = first_name;
    }
}
```

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
}  
public String getLastName() {  
    return lastName;  
}  
public void setLastName( String last_name ) {  
    this.lastName = last_name;  
}  
public int getSalary() {  
    return salary;  
}  
public void setSalary( int salary ) {  
    this.salary = salary;  
}  
}
```

There would be one table corresponding to each object you are willing to provide persistence. Consider above objects need to be stored and retrieved into the following RDBMS table:

```
create table EMPLOYEE (  
    id INT NOT NULL auto_increment,  
    first_name VARCHAR(20) default NULL,  
    last_name VARCHAR(20) default NULL,  
    salary INT default NULL,  
    PRIMARY KEY (id));
```

Based on the two above entities we can define following mapping file which instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-mapping PUBLIC  
"-//Hibernate/Hibernate Mapping DTD//EN"  
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
    <class name="Employee" table="EMPLOYEE">  
        <meta attribute="class-description">  
            This class contains the employee detail.  
        </meta>  
        <id name="id" type="int" column="id">  
            <generator class="native"/>  
        </id>  
        <property name="firstName" column="first_name" type="string"/>  
        <property name="lastName" column="last_name" type="string"/>  
        <property name="salary" column="salary" type="int"/>  
    </class>  
</hibernate-mapping>
```

You should save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml. Let us see little detail about the mapping elements used in the mapping file:

- The mapping document is an XML document having **<hibernate-mapping>** as the root element which contains all the <class> elements.

Advance Java Programming

- The **<class>** elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database table name is specified using the **table** attribute.
- The **<meta>** element is optional element and can be used to create the class description.
- The **<id>** element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.
- The **<generator>** element within the id element is used to automatically generate the primary key values. Set the **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity**, **sequence** or **hilo** algorithm to create primary key depending upon the capabilities of the underlying database.
- The **<property>** element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

Basic Example of Hibernate

Hibernate provide a way of mapping of applications class to the database table. It maps each class properties to the corresponding columns of table. After mapping to the table, all operations are performed by using the object of class in which table is mapped. Hibernate is an ORM based programming technology provides mapping library. Here is a simple example of mysql database connectivity using hibernate. Here we are describing this example step by step.

Step 1. Configuring Hibernate-

hibernate-cfg.xml is configuration file saved in the same folder where the source code of class file is saved. It creates the connection pool and set-up required environment.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>

    <!-- Database connection settings -->

    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/hibernate</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>
    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>
    <!-- Disable the second-level cache -->
    <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
    <!-- Echo all executed SQL to stdout -->
```

Advance Java Programming

```
<property name="show_sql">true</property>
<!-- Drop and re-create the database schema on startup -->
<property name="hbm2ddl.auto">none</property>
<mapping class="net.roseindia.table.Employee" />
</session-factory>
</hibernate-configuration>
```

In <hibernate-configuration> you configure your database connectivity, set sql dialect ,create database schema etc.

Here hibernate is your database name which contain tables. "net.roseindia.table.Employee" is your class in which you are mapping your table.

Step 2. Persistent class:

Now create Persistent class ?Hibernate uses the Plain Old Java Object (POJO) classes to map to the database table.

Here is the code of Employee.java-

```
package net.roseindia.table;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Entity
@Table(name = "employee") //database table name employee
public class Employee implements Serializable {
    @Id
    @GeneratedValue
    private int empId;
    @Column(name = "emp_name", nullable = false)
    private String empName;
    @Column(name = "emp_salary", nullable = false)
    private int salary;
    @Column(name = "designation", nullable = false)
    private String designation;
    @Column(name = "address", nullable = false)
    private String address;

    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public int getSalary() {
```


Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
return salary;
}
public void setSalary(int salary) {
this.salary = salary;
}
public String getDesignation() {
return designation;
}
public void setDesignation(String designation) {
this.designation = designation;
}
public String getAddress() {
return address;
}
public void setAddress(String address) {
this.address = address;}
}
```

Here we are using annotation for mapping our table employee to our class Employee.java.

Step3. Utility class: Now create an util class as HibernateUtil.java

```
package net.roseindia.util;
import org.hibernate.HibernateException;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
public class HibernateUtil {
    private static SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new AnnotationConfiguration().configure()
                .buildSessionFactory();

        } catch (HibernateException exception) {
            exception.printStackTrace();
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Step 4. Main Class: Now we are going to create our main class. Here we are inserting data in to employee table. session.save(employee) saves the employee object into the database table.

```
package net.roseindia.application;
import net.roseindia.table.Employee;
import net.roseindia.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
public class MainClaz {
```

Advance Java Programming

```
public static void main(String[] args) {  
    SessionFactory sessionFactory = HibernateUtil.getSessionFactory();  
    Session session = sessionFactory.openSession();  
    Transaction transaction = session.beginTransaction();  
    Employee employee = new Employee();  
    employee.setEmpName("Rose");  
    employee.setAddress("Patna");  
    employee.setSalary(18000);  
    employee.setDesignation("Manager");  
    session.save(employee);  
    transaction.commit();  
    session.clear();  
    session.close();  
    sessionFactory.close();  
}
```

output:

Hibernate: select employee0_.empId as empId0_0_, employee0_.address as address0_0_, employee0_.designation as designat3_0_0_, employee0_.emp_name as emp4_0_0_, employee0_.emp_salary as emp5_0_0_ from employee employee0_ where employee0_.empId=?

Hibernate: insert into employee (address, designation, emp_name, emp_salary) values (?, ?, ?, ?)

Annotation

Annotations in computer programming languages provide data about a program that is not part of the program itself by decorating them. They does not impact directly to the operation of the code to which they annotate.

Annotations are mainly used for:

- **Information for the compiler:** Compiler uses the annotations to detect errors or suppress warnings.
- **Compiler-time and deployment time processing:** annotation information is processed by the annotation processor in order to produce new source code and other files.
- **Runtime processing:** some annotations are scanned at runtime by using various techniques and open source libraries.

We can apply annotations to a program's declarations of classes, fields, methods, and other program elements. Rather than directly affecting on the program's semantic, they do affect the way programs are treated by tools and libraries, that can affect the semantics of an executing program. We can read the annotations from the class file, source files, or at run time.

Annotations are processed by compiler plug-ins called annotation processors whenever java source code is compiled. Processors are capable of producing informational messages, create additional Java source files or resources which in turn may be compiled and processed. But processors cannot modify the annotated code itself. The Java compiler stores annotation metadata in the class files only when the annotation has a RetentionPolicy of CLASS or RUNTIME, that is used to determine how to interact with the program elements or to change their behavior.

Hibernate Inheritance

Hibernate supports the three basic inheritance mapping strategies:

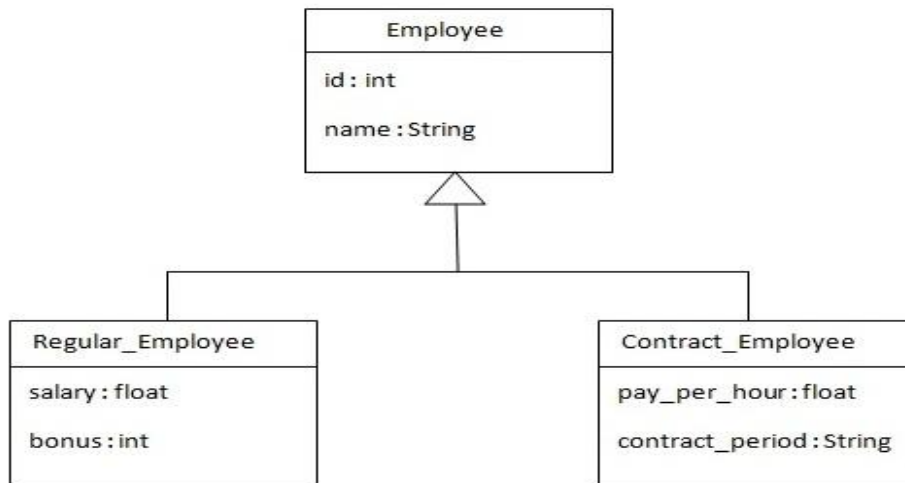
Table Per Hierarchy:

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table

Advance Java Programming

By this inheritance strategy, we can map the whole hierarchy by single table only. Here, an extra column (also known as discriminator column) is created in the table to identify the class.

Let's understand the problem first. I want to map the whole hierarchy given below into one table of the database.



There are three classes in this hierarchy. Employee is the super class for Regular_Employee and Contract_Employee classes.

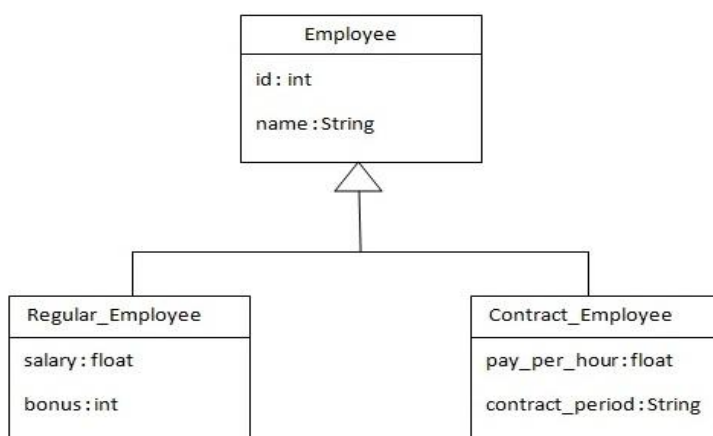
Table Per Concrete class

In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.

In case of Table Per Concrete class, there will be three tables in the database having no relations to each other. There are two ways to map the table with table per concrete class strategy.

- By union-subclass element
- By Self creating the table for each class

Let's understand what hierarchy we are going to map.



In case of table per concrete class, there will be three tables in the database, each representing a

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

particular class.

The union-subclass subelement of class, specifies the subclass. It adds the columns of parent table into this table. In other words, it is working as a union.

Table Per Subclass

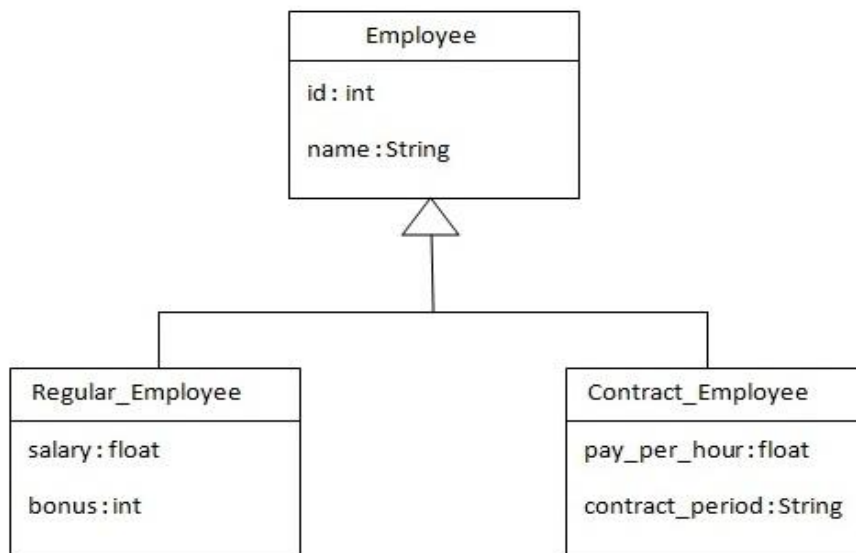
In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.

In case of Table Per Subclass, subclass mapped tables are related to parent class mapped table by primary key and foreign key relationship.

The **<joined-subclass>** element of class is used to map the child class with parent using the primary key and foreign key relation.

In this example, we are going to use `hibernate.hbm2ddl.auto` property to generate the table automatically. So we don't need to be worried about creating tables in the database.

Let's see the hierarchy of classes that we are going to map.



In case of table per subclass class, there will be three tables in the database, each representing a particular class.

The **joined-subclass** subelement of class, specifies the subclass. The **key** subelement of joined-subclass is used to generate the foreign key in the subclass mapped table. This foreign key will be associated with the primary key of parent class mapped table.

Inheritance Annotations

So far you have seen how Hibernate uses XML mapping file for the transformation of data from POJO to database tables and vice versa. Hibernate annotations is the newest way to define mappings without a use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata.

Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

If you going to make your application portable to other EJB 3 compliant ORM applications, you must use annotations to represent the mapping information but still if you want greater flexibility then you should go with XML-based mappings.

Environment Setup for Hibernate Annotation

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

First of all you would have to make sure that you are using JDK 5.0 otherwise you need to upgrade your JDK to JDK 5.0 to take advantage of the native support for annotations.

Second, you will need to install the Hibernate 3.x annotations distribution package, available from the sourceforge: (Download Hibernate Annotation) and copy hibernate-annotations.jar, lib/hibernate-comons-annotations.jar and lib/ejb3-persistence.jar from the Hibernate Annotations distribution to your CLASSPATH

Annotated Class Example:

As I mentioned above while working with Hibernate Annotation all the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

Consider we are going to use following EMPLOYEE table to store our objects:

```
create table EMPLOYEE (  
    id INT NOT NULL auto_increment,  
    first_name VARCHAR(20) default NULL,  
    last_name VARCHAR(20) default NULL,  
    salary INT default NULL,  
    PRIMARY KEY (id)  
);
```

Following is the mapping of Employee class with annotations to map objects with the defined EMPLOYEE table:

```
import javax.persistence.*;  
@Entity  
@Table(name = "EMPLOYEE")  
public class Employee {  
    @Id @GeneratedValue  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "first_name")  
    private String firstName;  
  
    @Column(name = "last_name")  
    private String lastName;  
  
    @Column(name = "salary")  
    private int salary;  
  
    public Employee() {}  
    public int getId() {  
        return id;  
    }  
    public void setId( int id ) {  
        this.id = id;  
    }  
    public String getFirstName() {  
        return firstName;  
    }  
    public void setFirstName( String first_name ) {
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
    this.firstName = first_name;
}
public String getLastName() {
    return lastName;
}
public void setLastName( String last_name ) {
    this.lastName = last_name;
}
public int getSalary() {
    return salary;
}
public void setSalary( int salary ) {
    this.salary = salary;
}
}
```

Hibernate detects that the `@Id` annotation is on a field and assumes that it should access properties on an object directly through fields at runtime. If you placed the `@Id` annotation on the `getId()` method, you would enable access to properties through getter and setter methods by default. Hence, all other annotations are also placed on either fields or getter methods, following the selected strategy. Following section will explain the annotations used in the above class.

@Entity Annotation:

The EJB 3 standard annotations are contained in the `javax.persistence` package, so we import this package as the first step. Second we used the `@Entity` annotation to the `Employee` class which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

@Table Annotation:

The `@Table` annotation allows you to specify the details of the table that will be used to persist the entity in the database.

The `@Table` annotation provides four attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table. For now we are using just table name which is `EMPLOYEE`.

@Id and @GeneratedValue Annotations:

Each entity bean will have a primary key, which you annotate on the class with the `@Id` annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.

By default, the `@Id` annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the `@GeneratedValue` annotation which takes two parameters strategy and generator which I'm not going to discuss here, so let us use only default the default key generation strategy. Letting Hibernate determine which generator type to use makes your code portable between different databases.

@Column Annotation:

The `@Column` annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes:

- name attribute permits the name of the column to be explicitly specified.
- length attribute permits the size of the column used to map a value particularly for a String value.
- nullable attribute permits the column to be marked NOT NULL when the schema is generated.
- unique attribute permits the column to be marked as containing only unique values.

Create Application Class:

Advance Java Programming

Finally, we will create our application class with the main() method to run the application. We will use this application to save few Employee's records and then we will apply CRUD operations on those records.

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
    private static SessionFactory factory;
    public static void main(String[] args) {
        try{
            factory = new AnnotationConfiguration().
                configure().
                //addPackage("com.xyz") //add package if used.
                addAnnotatedClass(Employee.class).
                buildSessionFactory();
        }catch (Throwable ex) {
            System.err.println("Failed to create sessionFactory object." + ex);
            throw new ExceptionInInitializerError(ex);
        }
        ManageEmployee ME = new ManageEmployee();

        /* Add few employee records in database */
        Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
        Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
        Integer empID3 = ME.addEmployee("John", "Paul", 10000);

        /* List down all the employees */
        ME.listEmployees();

        /* Update employee's records */
        ME.updateEmployee(empID1, 5000);

        /* Delete an employee from the database */
        ME.deleteEmployee(empID2);

        /* List down new list of the employees */
        ME.listEmployees();
    }
    /* Method to CREATE an employee in the database */
    public Integer addEmployee(String fname, String lname, int salary){
        Session session = factory.openSession();
```

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
Transaction tx = null;
Integer employeeID = null;
try{
    tx = session.beginTransaction();
    Employee employee = new Employee();
    employee.setFirstName(fname);
    employee.setLastName(lname);
    employee.setSalary(salary);
    employeeID = (Integer) session.save(employee);
    tx.commit();
}catch (HibernateException e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
}finally {
    session.close();
}
return employeeID;
}
/* Method to READ all the employees */
public void listEmployees( ){
    Session session = factory.openSession();
    Transaction tx = null;
    try{
        tx = session.beginTransaction();
        List employees = session.createQuery("FROM Employee").list();
        for (Iterator iterator =
            employees.iterator(); iterator.hasNext();){
            Employee employee = (Employee) iterator.next();
            System.out.print("First Name: " + employee.getFirstName());
            System.out.print(" Last Name: " + employee.getLastName());
            System.out.println(" Salary: " + employee.getSalary());
        }
        tx.commit();
    }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
}
/* Method to UPDATE salary for an employee */
public void updateEmployee(Integer EmployeeID, int salary ){
    Session session = factory.openSession();
    Transaction tx = null;
    try{
        tx = session.beginTransaction();
        Employee employee =
            (Employee)session.get(Employee.class, EmployeeID);
```


Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
        employee.setSalary( salary );
        session.update(employee);
        tx.commit();
    }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
}
/* Method to DELETE an employee from the records */
public void deleteEmployee(Integer EmployeeID){
    Session session = factory.openSession();
    Transaction tx = null;
    try{
        tx = session.beginTransaction();
        Employee employee =
            (Employee)session.get(Employee.class, EmployeeID);
        session.delete(employee);
        tx.commit();
    }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
}
}
```

Database Configuration:

Now let us create hibernate.cfg.xml configuration file to define database related parameters. This time we are not going

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property name="hibernate.connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <!-- Assume students is the database name -->
        <property name="hibernate.connection.url">
            jdbc:mysql://localhost/test
        </property>
        <property name="hibernate.connection.username">
            root
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
</property>
<property name="hibernate.connection.password">
    cohondob
</property>
</session-factory>
</hibernate-configuration>
```

Compilation and Execution:

Here are the steps to compile and run the above mentioned application. Make sure you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

- Delete Employee.hbm.xml mapping file from the path.
- Create Employee.java source file as shown above and compile it.
- Create ManageEmployee.java source file as shown above and compile it.
- Execute ManageEmployee binary to run the program.

You would get following result, and records would be created in EMPLOYEE table.

\$java ManageEmployee

.....VARIOUS LOG MESSAGES WILL DISPLAY HERE.....

First Name: Zara Last Name: Ali Salary: 1000

First Name: Daisy Last Name: Das Salary: 5000

First Name: John Last Name: Paul Salary: 10000

First Name: Zara Last Name: Ali Salary: 5000

First Name: John Last Name: Paul Salary: 10000

If you check your EMPLOYEE table, it should have following records:

mysql> select * from EMPLOYEE;

```
+---+-----+-----+-----+
| id | first_name | last_name | salary |
+---+-----+-----+-----+
| 29 | Zara      | Ali      | 5000   |
| 31 | John      | Paul     | 10000  |
+---+-----+-----+-----+
```

2 rows in set (0.00 sec)

mysql>

Hibernate Sessions

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed. The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of the following three states at a given point in time:

- **transient:** A new instance of a a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
- **persistent:** You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.
- **detached:** Once we close the Hibernate Session, the persistent instance will become a detached instance.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

A Session instance is serializable if its persistent classes are serializable. A typical transaction should use the following idiom:

```
Session session = factory.openSession();
Transaction tx = null;
try {
    tx = session.beginTransaction();
    // do some work
    ...
    tx.commit();
}
catch (Exception e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
}finally {
    session.close();
}
```

If the Session throws an exception, the transaction must be rolled back and the session must be discarded.

Session Interface Methods:

There are number of methods provided by the **Session** interface but I'm going to list down few important methods only, which we will use in this tutorial. You can check Hibernate documentation for a complete list of methods associated with **Session** and **SessionFactory**.

S.N. Session Methods and Description

- 1 **Transaction beginTransaction()**
Begin a unit of work and return the associated Transaction object.
- 2 **void cancelQuery()**
Cancel the execution of the current query.
- 3 **void clear()**
Completely clear the session.
- 4 **Connection close()**
End the session by releasing the JDBC connection and cleaning up.
- 5 **Criteria createCriteria(Class persistentClass)**
Create a new Criteria instance, for the given entity class, or a superclass of an entity class.
- 6 **Criteria createCriteria(String entityName)**
Create a new Criteria instance, for the given entity name.
- 7 **Serializable getIdentifier(Object object)**
Return the identifier value of the given entity as associated with this session.
- 8 **Query createFilter(Object collection, String queryString)**
Create a new instance of Query for the given collection and filter string.
- 9 **Query createQuery(String queryString)**
Create a new instance of Query for the given HQL query string.
- 10 **SQLQuery createSQLQuery(String queryString)**
Create a new instance of SQLQuery for the given SQL query string.
- 11 **void delete(Object object)**
Remove a persistent instance from the datastore.

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

- 12 **void delete(String entityName, Object object)**
Remove a persistent instance from the datastore.
- 13 **Session get(String entityName, Serializable id)**
Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.
- 14 **SessionFactory getSessionFactory()**
Get the session factory which created this session.
- 15 **void refresh(Object object)**
Re-read the state of the given instance from the underlying database.
- 16 **Transaction getTransaction()**
Get the Transaction instance associated with this session.
- 17 **boolean isConnected()**
Check if the session is currently connected.
- 18 **boolean isDirty()**
Does this session contain any changes which must be synchronized with the database?
- 19 **boolean isOpen()**
Check if the session is still open.
- 20 **Serializable save(Object object)**
Persist the given transient instance, first assigning a generated identifier.
- 21 **void saveOrUpdate(Object object)**
Either save(Object) or update(Object) the given instance.
- 22 **void update(Object object)**
Update the persistent instance with the identifier of the given detached instance.
- 23 **void update(String entityName, Object object)**
Update the persistent instance with the identifier of the given detached instance.

Introduction to Spring framework

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, reusable code.

Spring framework is an open source Java platform and it was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

The Spring framework is an Open Source framework. It provides the light-weight framework to develop maintainable and reusable enterprise applications. It provides all the services of enterprise application using programmatic and declaratively.

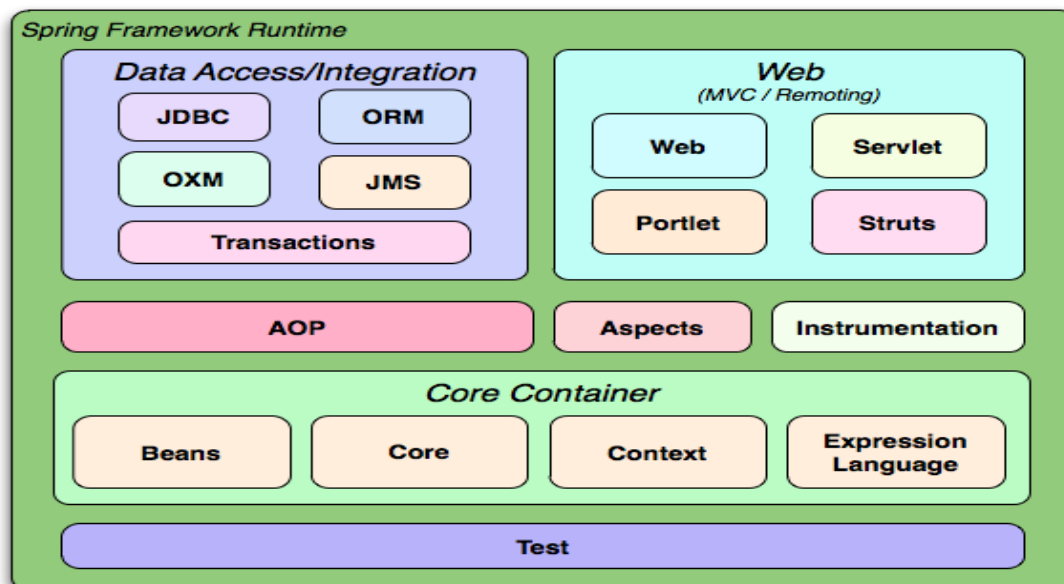
It works with all tier software application layers. It is non-invasive (does not force to extend any class or implement any interface) and handles the infrastructure so developer only focuses on their business logic. It is used to develop maintainable and reusable enterprise applications.

It provides very simple and rich facilities to integrate various frameworks technologies, and services in the application. Spring enables you to enjoy the key benefits of J2EE, while minimizing the complexity encountered by application code. Developed in 2004 by Rod Johnson, an experienced J2EE architect.

Spring Architecture

The Spring framework provides one-stop shop for java based application on all layers (one tier- stand alone java application, web tier- in web application and enterprise tier- Enterprise Java Beans). It is modular, means choose spring module based on requirements, It does not inforce to add all the library files in your project classpath. Here are complete module details of the Spring Framework

All the features of Spring framework are organized into 20 modules. The diagrammatic architecture as follows :



Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Spring Core : It is core part of Spring and consists of the following modules – Core, Beans, Context and Expression Language. The brief description is as follows

- **Core :** It is fundamental module of the framework with IOC and Dependency Injection with singleton design pattern.
- **Beans :** This module is implementation of the factory design pattern through BeanFactory. The BeanFactory applies IOC to separate the application's configuration and dependency specification from actual program logic.
- **Context :** It (ApplicationContext) extends the concept of BeanFactory, adding support for - Internationalization (I18N) messages, Application lifecycle events and Validation. Also includes Enterprise services such as E-mail, JNDI access, EJB integration, Remoting, and Scheduling.
- **Expression Language :** The Spring3.0 introduces a new expression language – Spring Expression Language (SpEL). It is a powerful expression language based on Java Server Pages (JSP) Expression Language(EL). It is used to write expression language querying various beans, accessing and manipulating their properties and invoking the methods.

Data Access : It is fundamental part of database access layer and consists of the following modules – JDBC, ORM, OXM, JMS and Transaction management module.

The brief description is as follows :

- **JDBC:** The JDBC modules provides a JDBC-abstraction layer that removes the complexity of the traditional JDBC code and parsing of database-vendor specific error code.
- **ORM:** The ORM module provide consistency/portability to your code regardless of data access technologies based on object oriented mapping concept like Hibernate, JPA, JDO and iBatis. It provides code without worrying about catching exceptions specific to each persistence technology (ex: SQLException thrown by JDBC API).
- **OXM:** The OXM introduces in Spring3.0 as separate module. It is used to converts object into XML format and vice versa. The Spring OXM provides a uniform API to access any of these OXM(Castor, XStream, JiBX, Java API for XML and XmlBeans) framework.
- **JMS :** The JMS module provides by reducing the number of line of code to send and receive messages. The API take car of JMS workflow and exception handling.
- **Transaction :** The Transaction module supports programmatic and declarative transaction management for POJO classes. All the enterprise level transaction implementation concepts can be implement in Spring.

Web : It is core part of Web layer and consists of the following modules – Web, Web-Servlet, Web-Struts and Web-Portlet. The brief description is as follows :

- **Web :** This module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- **Web-Servlet :** The Web-Servlet module contains model-view-controller (MVC) based implementation for web applications. It provides all other features of MVC including UI tags and data validations.
- **Web-Struts:** The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application. It contains the classes to integrate Struts1.x and Struts2.
- **Web-Portlet :** The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Others : There are few other important modules in Spring, which plays vital role in the framework to use all the features in various scenario. The modules are AOP, Aspect, Instrumentation, and Test.

- **AOP :** It contains API for AOP Alliance-complaint aspect-oriented programming implementations on various layers. You can introduce new functionalities into existing code without modifying it.
- **Aspectj :** The separate Aspects module provides integration with AspectJ.
- **Test :** The Instrumentation module provides class instrumentation support and classloader implementations to be used in certain application servers.

Spring framework definition

The Spring Framework is an open source application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform.

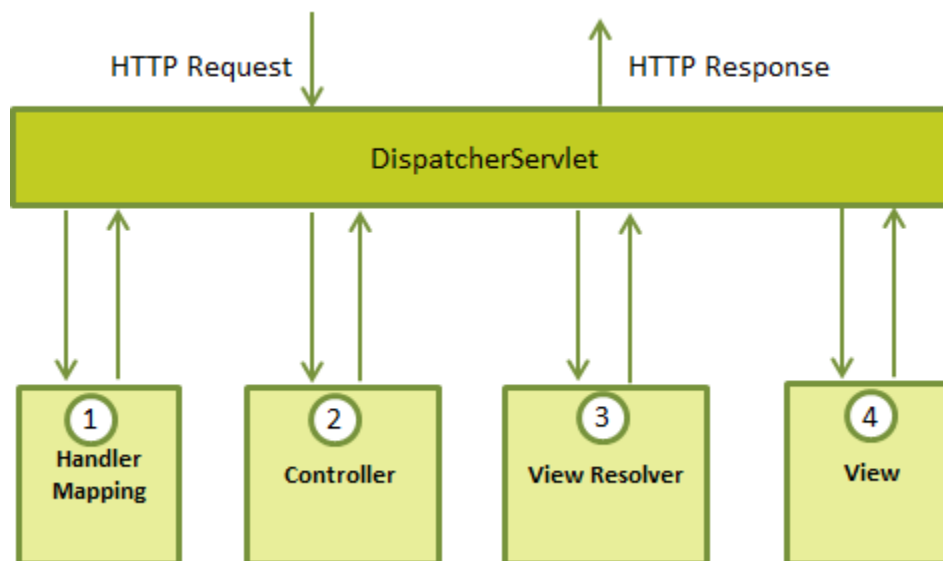
Spring and MVC

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

- The **Model** encapsulates the application data and in general they will consist of POJO.
- The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The **Controller** is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC *DispatcherServlet* is illustrated in the following diagram:



Following is the sequence of events corresponding to an incoming HTTP request to *DispatcherServlet*:

1. After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.

Advance Java Programming

2. The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
3. The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
4. Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

All the above mentioned components ie. HandlerMapping, Controller and ViewResolver are parts of *WebApplicationContext* which is an extension of the plain *ApplicationContext* with some extra features necessary for web applications.

Required Configuration

You need to map requests that you want the *DispatcherServlet* to handle, by using a URL mapping in the **web.xml** file. The following is an example to show declaration and mapping for **HelloWeb** *DispatcherServlet* example:

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Spring MVC Application</display-name>
  <servlet>
    <servlet-name>HelloWeb</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWeb</servlet-name>
    <url-pattern>*.jsp</url-pattern>
  </servlet-mapping>
</web-app>
```

The **web.xml** file will be kept *WebContent/WEB-INF* directory of your web application. OK, upon initialization of **HelloWeb** *DispatcherServlet*, the framework will try to load the application context from a file named **[servlet-name]-servlet.xml** located in the application's *WebContent/WEB-INF* directory. In this case our file will be **HelloWeb-servlet.xml**.

Next, `<servlet-mapping>` tag indicates what URLs will be handled by the which *DispatcherServlet*. Here all the HTTP requests ending with **.jsp** will be handled by the **HelloWeb** *DispatcherServlet*.

If you do not want to go with default filename as *[servlet-name]-servlet.xml* and default location as *WebContent/WEB-INF*, you can customize this file name and location by adding the servlet listener *ContextLoaderListener* in your web.xml file as follows:

```
<web-app...>
<!-- DispatcherServlet definition goes here -->
....
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>
</context-param>
```


Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
</web-app>
```

Now, let us check the required configuration for **HelloWeb-servlet.xml** file, placed in your web application's *WebContent/WEB-INF* directory:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <context:component-scan base-package="com.tutorialspoint" />
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

Following are the important points about **HelloWeb-servlet.xml** file:

- The *[servlet-name]-servlet.xml* file will be used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.
- The *<context:component-scan...>* tag will be use to activate Spring MVC annotation scanning capability which allows to make use of annotations like *@Controller* and *@RequestMapping* etc.
- The *InternalResourceViewResolver* will have rules defined to resolve the view names. As per the above defined rule, a logical view named **hello** is delegated to a view implementation located at */WEB-INF/jsp/hello.jsp* .

Next section will show you how to create your actual components ie. Controller, Model and View.

Defining a Controller

DispatcherServlet delegates the request to the controllers to execute the functionality specific to it. The **@Controller** annotation indicates that a particular class serves the role of a controller. The **@RequestMapping** annotation is used to map a URL to either an entire class or a particular handler method.

```
@Controller
@RequestMapping("/hello")
public class HelloController{

    @RequestMapping(method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Advance Java Programming

The **@Controller** annotation defines the class as a Spring MVC controller. Here, the first usage of **@RequestMapping** indicates that all handling methods on this controller are relative to the **/hello** path. Next annotation **@RequestMapping(method = RequestMethod.GET)** is used to declare the *printHello()* method as the controller's default service method to handle HTTP GET request. You can define another method to handle any POST request at the same URL.

You can write above controller in another form where you can add additional attributes in **@RequestMapping** as follows:

@Controller

```
public class HelloController{
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

The **value** attribute indicates the URL to which the handler method is mapped and the **method** attribute defines the service method to handle HTTP GET request. There are following important points to be noted about the controller defined above:

- You will defined required business logic inside a service method. You can call another methods inside this method as per requirement.
- Based on the business logic defined, you will create a **model** within this method. You can setter different model attributes and these attributes will be accessed by the view to present the final result. This example creates a model with its attribute "message".
- A defined service method can return a String which contains the name of the **view** to be used to render the model. This example returns "hello" as logical view name.

Creating JSP Views

Spring MVC supports many types of views for different presentation technologies. These include - JSPs, HTML, PDF, Excel worksheets, XML, Velocity templates, XSLT, JSON, Atom and RSS feeds, JasperReports etc. But most commonly we use JSP templates written with JSTL. So let us write a simple **hello** view in **/WEB-INF/hello/hello.jsp**:

```
<html>
<head>
<title>Hello Spring MVC</title>
</head>
<body>
<h2>${message}</h2>
</body>
</html>
```

Here **\${message}** is the attribute which we have setup inside the Controller. You can have multiple attributes to be displayed inside your view.

Spring Context Definition:

The Application Context is spring's more advanced container. Similar to BeanFactory it can load bean definitions, wire beans together and dispense beans upon request. Additionally it adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the **org.springframework.context.ApplicationContext** interface.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

The ApplicationContext includes all functionality of the BeanFactory, it is generally recommended over the BeanFactory. BeanFactory can still be used for light weight applications like mobile devices or applet based applications.

The most commonly used ApplicationContext implementations are:

- FileSystemXmlApplicationContext: This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.
- ClassPathXmlApplicationContext This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.
- WebXmlApplicationContext: This container loads the XML file with definitions of all beans from within a web application.

We already have seen an example on ClassPathXmlApplicationContext container in Spring Hello World Example, and we will talk more about XmlWebApplicationContext in a separate chapter when we will discuss web based Spring applications. So let see one example on FileSystemXmlApplicationContext.

Example:

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step Description

- 1 Create a project with a name SpringExample and create a package com.tutorialspoint under the src folder in the created project.
- 2 Add required Spring libraries using Add External JARs option as explained in the Spring Hello World Example chapter.
- 3 Create Java classes HelloWorld and MainApp under the com.tutorialspoint package.
- 4 Create Beans configuration file Beans.xml under the src folder.
- 5 The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of HelloWorld.java file:

```
package com.tutorialspoint;
```

```
public class HelloWorld {  
    private String message;  
  
    public void setMessage(String message){  
        this.message = message;  
    }  
  
    public void getMessage(){  
        System.out.println("Your Message : " + message);  
    }  
}
```

Following is the content of the second file MainApp.java:

```
package com.tutorialspoint;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.FileSystemXmlApplicationContext;  
public class MainApp {  
    public static void main(String[] args) {  
        ApplicationContext context = new FileSystemXmlApplicationContext
```

Advance Java Programming

```
("C:/Users/ZARA/workspace/HelloSpring/src/Beans.xml");
HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
obj.getMessage();
}
}
```

There are following two important points to note about the main program:

1. First step is to create factory object where we used framework API `FileSystemXmlApplicationContext` to create the factory bean after loading the bean configuration file from the given path. The `FileSystemXmlApplicationContext()` API takes care of creating and initializing all the objects ie. beans mentioned in the XML bean configuration file.
2. Second step is used to get required bean using `getBean()` method of the created context. This method uses bean ID to return a generic object which finally can be casted to actual object. Once you have object, you can use this object to call any class method.

Following is the content of the bean configuration file Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
    <property name="message" value="Hello World!"/>
  </bean>
</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

Your Message : Hello World!

Inversion of Control (IOC) in spring

The basic concept of the dependency injection (also known as Inversion of Control pattern) is that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (in the case of the Spring framework, the IOC container) is then responsible for hooking it all up.

i.e., Applying IoC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is, dependencies are injected into objects. So, IoC means an inversion of responsibility with regard to how an object obtains references to collaborating objects.

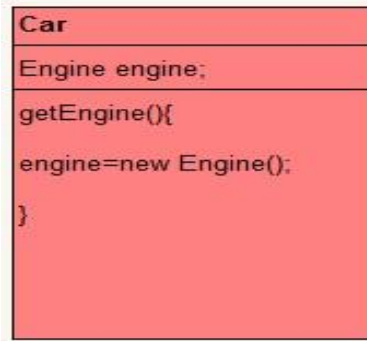
Example

Lets we have two classes-Car and Engine.Car has a object of Engine.

Normal way:

There are many ways to instantiate a object. A simple and common way is with new operator. so here Car class contain object of Engine and we have it instantiated using new operator.

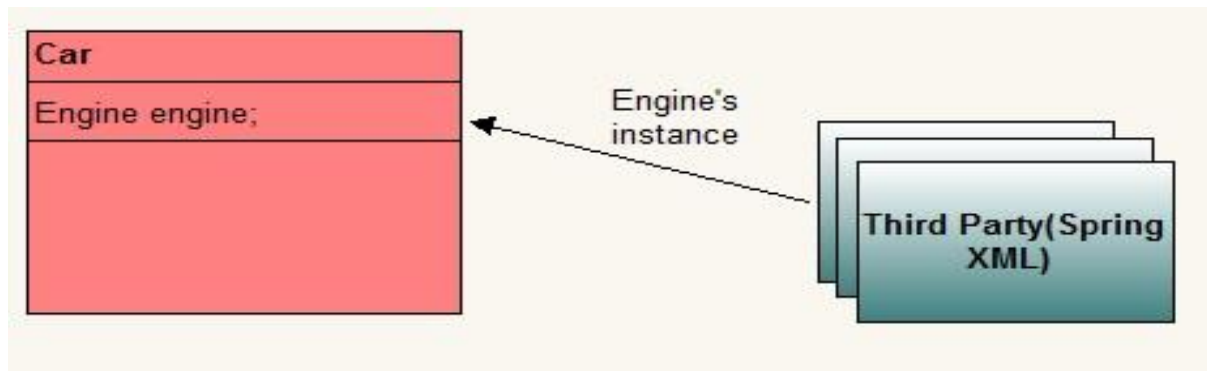
Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming



Without DI

With help of Dependency Injection:

Now we outsource instantiation and supply job of instantiating to third party. **Car** needs object of **Engine** to operate but it outsources that job to some third party. The designated third party, decides the moment of instantiation and the type to use to create the instance. The dependency between class **Car** and class **Engine** is injected by a third party. Whole of this agreement involves some configuration information too. This whole process is called dependency injection.



With DI

How this whole dependency injection works, we will see it in further posts.

Benefits of Dependency Injection in Spring:

- Ensures configuration and uses of services are separate.
- Can switch implementations by just changing configuration.
- Enhances Testability as mock dependencies can be injected.
- Dependencies can be easily identified.
- No need to read code to see what dependencies your code talks to.

Types of Dependency Injection:

- **Setter Injection:** Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.
- **Constructor Injection:** Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.
- **Interface Injection:** In interface-based dependency injection, we will have an interface and on implementing it we will get the instance injected.

Advance Java Programming

Aspect Oriented programming in Spring (AOP)

One of the key components of Spring Framework is the **Aspect oriented programming (AOP)** framework. Aspect Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called **cross-cutting concerns** and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing,

. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect. AOP is like triggers in programming languages such as Perl, .NET, Java and others.

Spring AOP module provides interceptors to intercept an application, for example, when a method is executed, you can add extra functionality before or after the method execution.

AOP Terminologies:

Before we start working with AOP, let us become familiar with the AOP concepts and terminology. These terms are not specific to Spring, rather they are related to AOP.

Terms	Description
Aspect	A module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement.
Join point	This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.
Advice	This is the actual action to be taken either before or after the method execution. This is actual piece of code that is invoked during program execution by Spring AOP framework.
Pointcut	This is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.
Introduction	An introduction allows you to add new methods or attributes to existing classes.
Target object	The object being advised by one or more aspects, this object will always be a proxied object. Also referred to as the advised object.
Weaving	Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime.

Types of Advice

Spring aspects can work with five kinds of advice mentioned below:

Advice	Description
before	Run advice before the a method execution.
after	Run advice after the a method execution regardless of its outcome.
after-returning	Run advice after the a method execution only if method completes successfully.
after-throwing	Run advice after the a method execution only if method exits by throwing an

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

exception.

around

Run advice before and after the advised method is invoked.

Custom Aspects Implementation

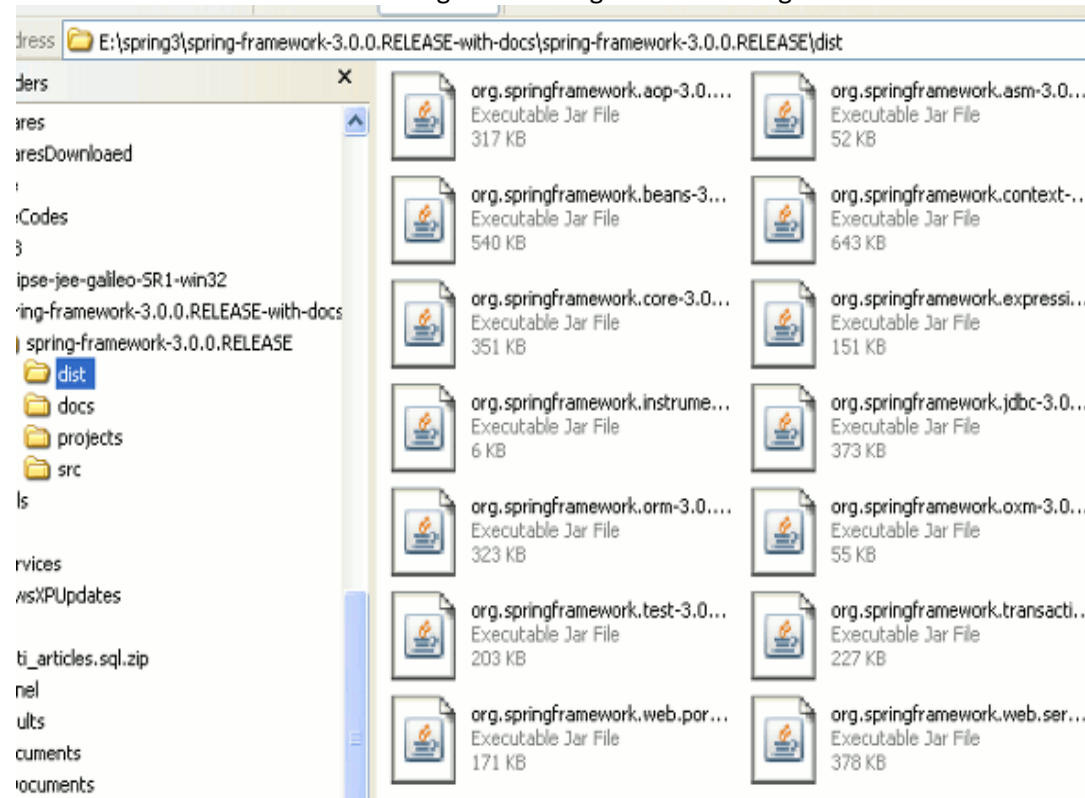
Spring supports the **@AspectJ annotation style** approach and the **schema-based** approach to implement custom aspects. These two approaches have been explained in detail in the following two sub chapters

Approach	Description
XML Schema based	Aspects are implemented using regular classes along with XML based configuration.
@AspectJ based	@AspectJ refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

Steps for Spring Application:

Step 1:

Download the latest version of Spring 3 from <http://www.springsource.org/download>. For this tutorial we have downloaded **spring-framework-3.0.0.RELEASE-with-docs.zip**, which contains the documentation also. After extracting the file we got the following directories:

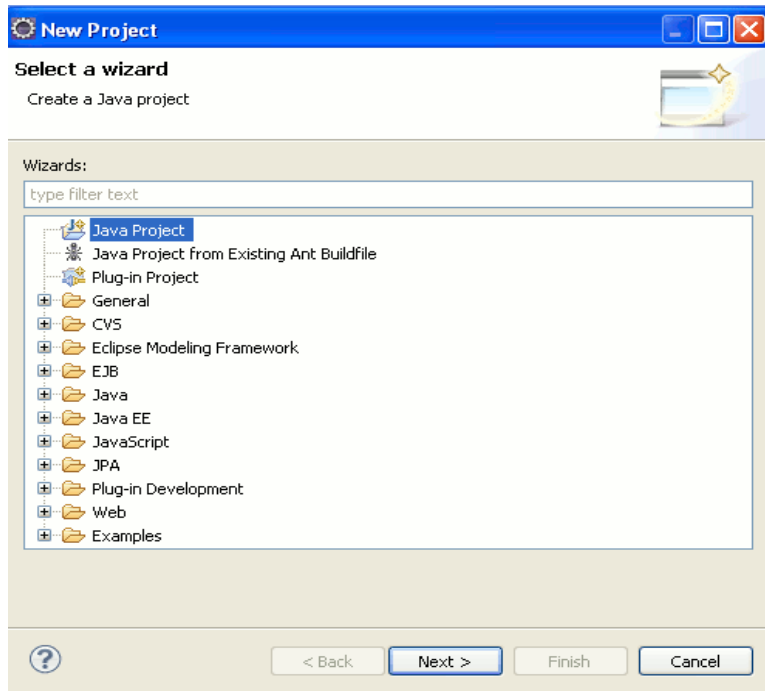


Step 2:

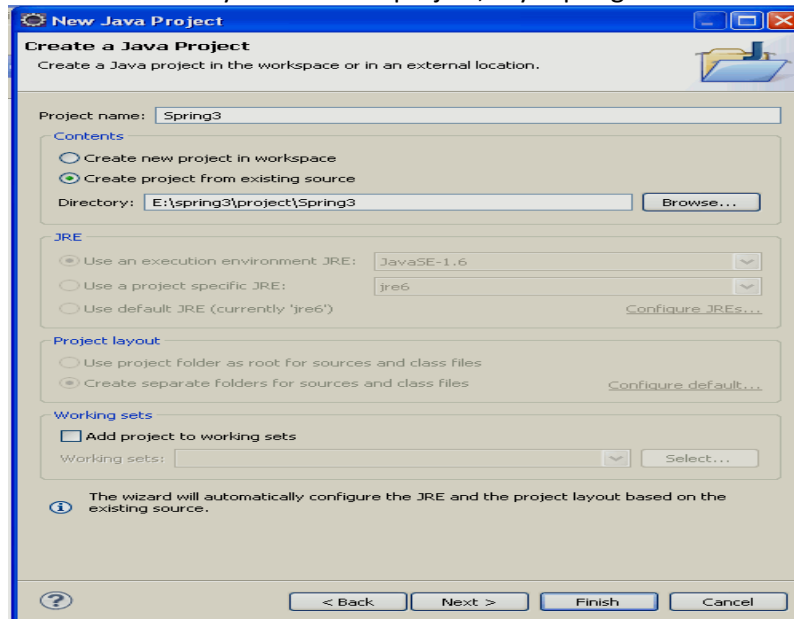
Now we will create a new project in Eclipse IDE and then add the library files.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming



Click next. Give any name to the project, say "Spring 3" and then click on the "Finish" button.



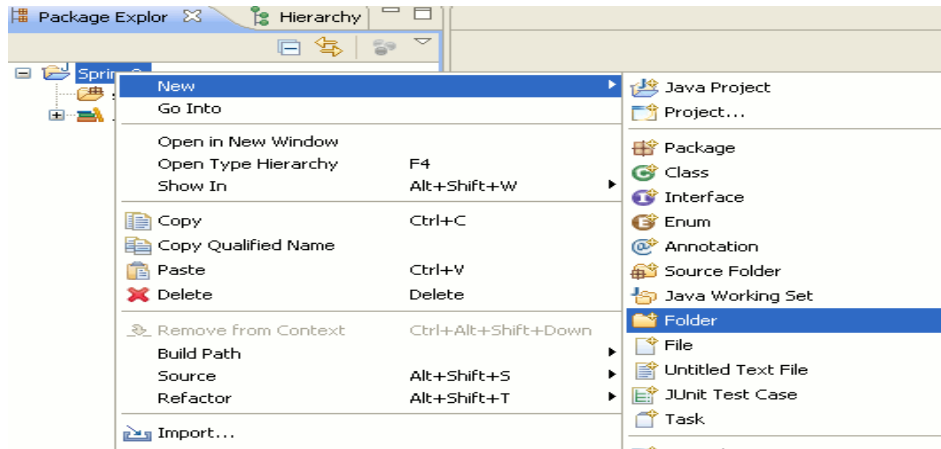
Eclipse will create a new project.

Step 3:

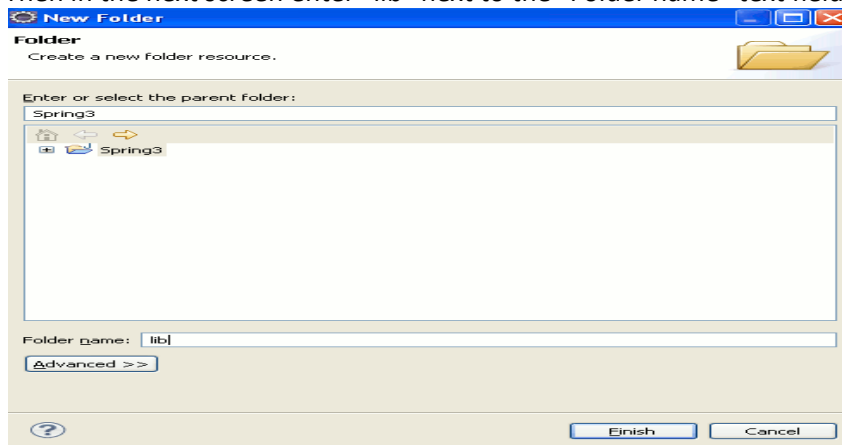
Create a new folder "lib" in the project space to hold the Spring 3.0 libraries. Right click on the "Spring3" in the project explorer and then select new folder option as shown below:

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

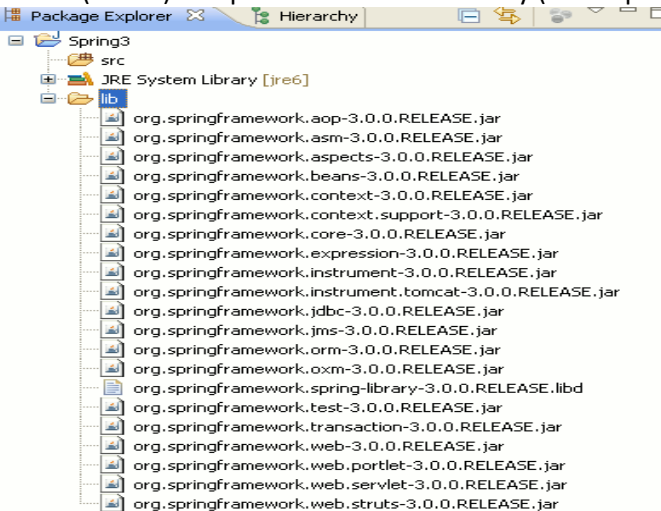


Then in the next screen enter "lib" next to the "Folder name" text field and click on the "Finish" button.



Step 4:

Now we will add the Spring 3. libraries to the project. Extract the "spring-framework-3.0.0.RELEASE-with-docs.zip" file if you have not extracted. Now go to the "dist" directory of the and then copy all the jar files (Ctrl+C) and paste on the **lib** directory (of our project) in the Eclipse IDE.



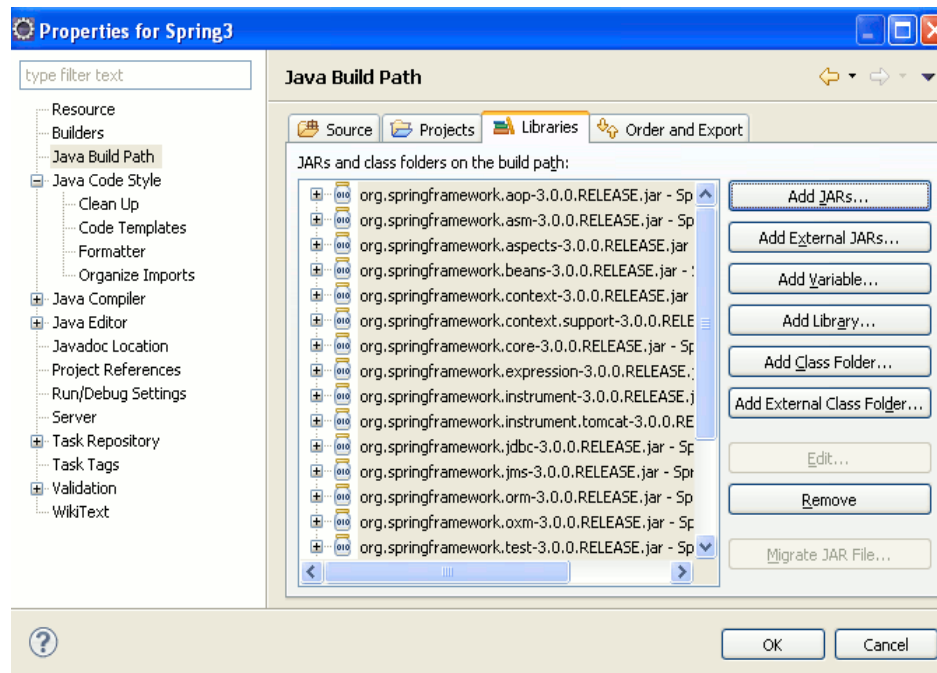
Then find the commons-logging.jar from extracted folder and also copy this file into Eclipse IDE. You will find this library into spring-framework-3.0.0.RELEASE-with-docs\spring-framework-3.0.0.RELEASE\projects\spring-build\lib\ivy folder.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

Step 5:

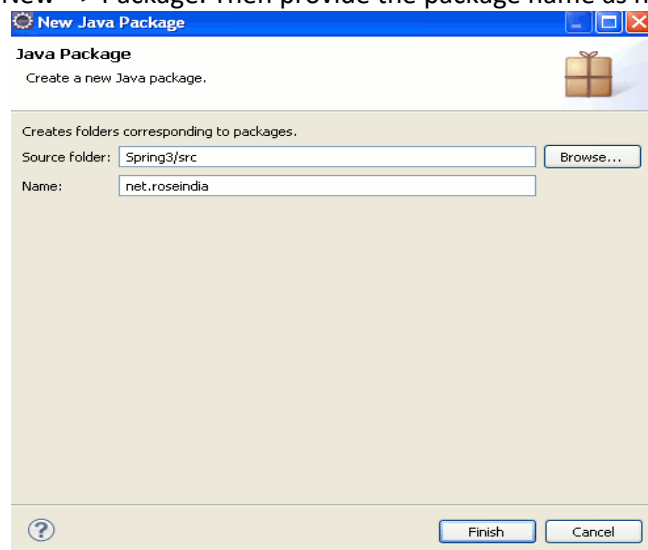
Now add all the libraries to "Java Build Path". Right click on the "Spring3" in project explorer and then select properties. Then select "Java Build Path" --> Libraries and then click on the "Add JARs" button. And add all the libraries to Java Build Path.



Then click on the "OK" button. This will add all the libraries to the project. Now we can proceed with our Spring 3 Hello World example.

Step 6:

Create a new package net.roseindia to hold the java files. Right click on the "Spring3" and then select New --> Package. Then provide the package name as net.roseindia and click on the "Finish" button.



Step 7:

Create a new Java file **Spring3HelloWorld.java** under the package **net.roseindia** and add the following code:

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

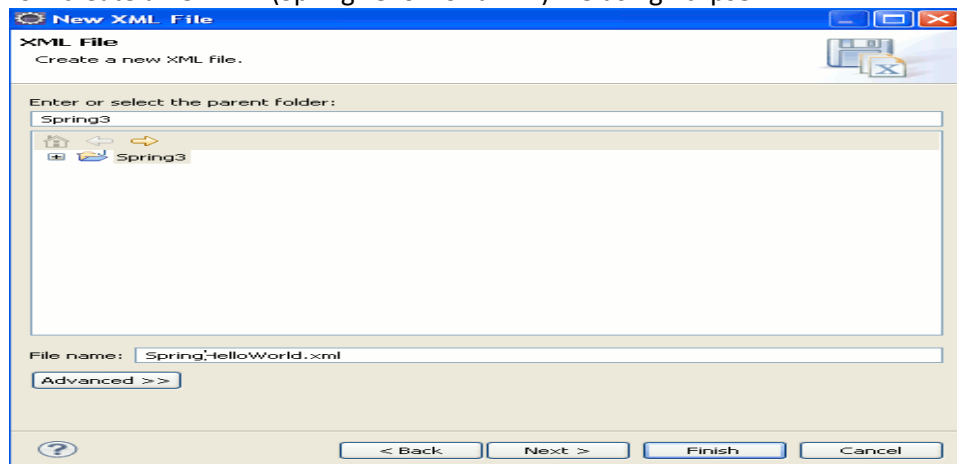
```
package net.bcaty;
```

```
public class Spring3HelloWorld {  
    public void sayHello(){  
        System.out.println("Hello Spring 3.0");  
    }  
}
```

In the above class we have created a method sayHello() which prints the "Hello Spring 3.0" on the console. In this section we will use the Spring framework to manage the Spring3HelloWorld bean, and then get the bean from the Spring runtime environment (Spring context) and then call the sayHello() method. In the next step we will create an xml file which will be used as Metadata to configure the bean.

Step 8:

Now create a new xml (SpringHelloWorld.xml) file using Eclipse IDE.



Add the following code to the xml file:

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:aop="http://www.springframework.org/schema/aop"  
    xsi:schemaLocation="http://www.springframework.org/schema/beans  
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
    <bean id="Spring3HelloWorldBean"  
        class="net.bcaty.Spring3HelloWorld" />  
    </beans>
```

The above xml file declares the spring bean "Spring3HelloWorldBean" of the class **net.bcaty.Spring3HelloWorld**. The <bean .../> tag is used to declare a bean in the xml file. Spring uses the xml file to configure the spring run-time environment. Spring framework manages the beans in our program. In the next sections we will learn Spring core components in detail. **Note: You should move the SpringHelloWorld.xml file into src directory of the project. Just use mouse to drag and drop in the src folder.**

Step 9:

Now create a java file (Spring3HelloWorldTest.java) into net.roseindia package and add the following code:

```
package net.bcaty;  
import java.util.Map;  
import org.springframework.beans.factory.xml.XmlBeanFactory;
```

Advance Java Programming

```
import org.springframework.core.io.ClassPathResource;
import org.springframework.util.Assert;
public class Spring3HelloWorldTest {
    public static void main(String[] args) {
        XmlBeanFactory beanFactory = new XmlBeanFactory(new ClassPathResource(
            "SpringHelloWorld.xml"));
        Spring3HelloWorld myBean = (Spring3HelloWorld) beanFactory
            .getBean("Spring3HelloWorldBean");
        myBean.sayHello();
    }
}
```

In the above code we have created the instance of **XmlBeanFactory** and the retrieved the "Spring3HelloWorldBean". Then we can call the sayHello() method on the bean. The **XmlBeanFactory** class is extension of DefaultListableBeanFactory that reads bean definitions from an XML document. In our case it reads the bean definitions from SpringHelloWorld.xml file.

Step 10:

To run the code in Eclipse open Spring3HelloWorldTest.java in the editor and then right click and select Run as --> Java Application. This execute the Spring3HelloWorldTest.java file and following output will be displayed in the console.

Sep 10, 2014 6:49:57 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [SpringHelloWorld.xml]

Hello Spring 3.0

Struts

Understanding Struts Framework:

Apache Struts is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture. It was originally created by Craig McClanahan and donated to the Apache Foundation in May, 2000.

Overview of the MVC design pattern

The model-view-controller design pattern, also known as Model 2 in J2EE application programming, is a well-established design pattern for programming. You use Struts to help you develop applications that are divided into three functional areas:

Model

The model is the business logic, which in most cases involves access of data stores like relational databases. The development team that handles the model may be expert at writing DB2® COBOL programs, or EJB entity beans, or some other technology appropriate for storing and manipulating enterprise data.

View

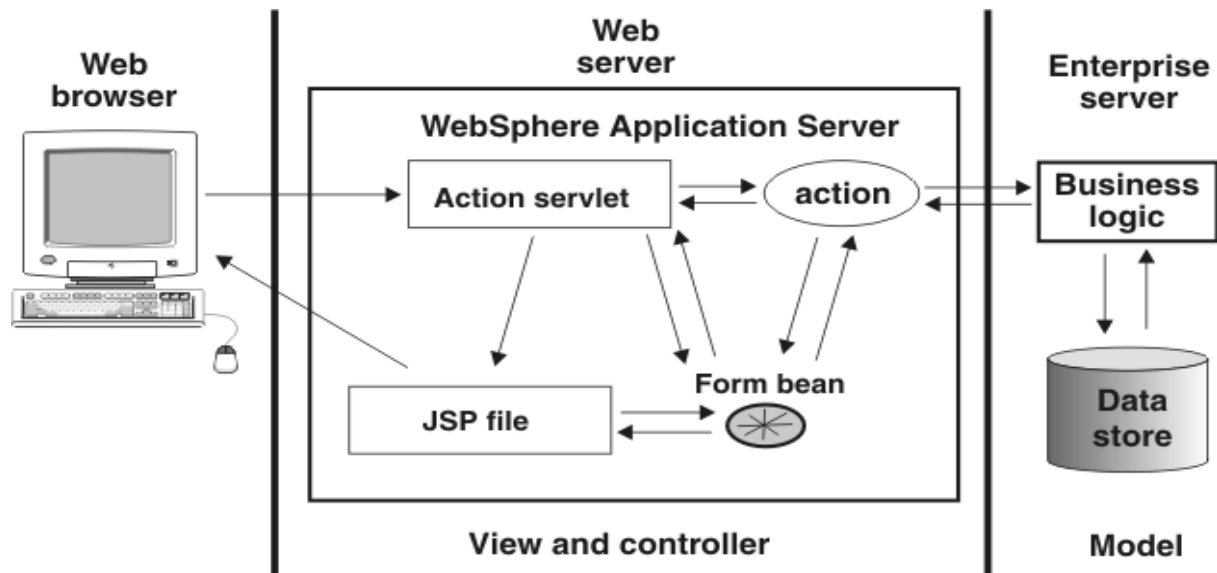
The view is the code that presents images and data on Web pages. The code comprises JSP pages and the Java beans that store data for use by the JSP pages.

Controller

The controller is the code that determines the overall flow of the application. It comprises one or more Struts actions, configuration files, and servlets.

As illustrated in the following diagram, the Web server at run time contains both the view and controller components of a Model 2 Web application, while a third tier (which is usually outside of the Web server)

contains the model. The diagram also references Struts-specific components that are described in the next section.



Characteristics of the MVC design pattern

Several characteristics are common to the model-view-controller design pattern.

View

The user interface generally is created with JSP files that do not themselves contain any business logic. These pages represent the view component of an MVC architecture.

Controller

Forms and hyperlinks in the user interface that require business logic to be executed are submitted to a request URI that is mapped to an action servlet. One instance of this servlet class exists and receives and processes all requests that change the state of a user's interaction with the application. This component represents the controller component of an MVC architecture.

Model

The action servlet selects and invokes one or more actions to perform the requested business logic. The actions manipulate the state of the application's interaction with the user, typically by creating or modifying Java beans that are stored as request or session attributes (depending on how long they need to be available). Such Java beans represent the model component of an MVC architecture. Instead of producing the next page of the user interface directly, actions generally use the `RequestDispatcher.forward()` facility of the servlet API to pass control to an appropriate JSP file to produce the next page of the user interface.

View, controller, and Struts

The Struts contribution to the view component of the MVC design pattern is twofold:

- Struts provides the Java class `org.apache.struts.action.ActionForm`, which a Java developer subclasses to create a form bean. At run time, the bean is used in two ways:
 - When a JSP page prepares the related HTML form for display, the JSP page accesses the bean, which holds values to be placed into the form. Those values are provided from business logic or from previous user input.

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

- When user input is returned from a Web browser, the bean validates and holds that input either for use by business logic or (if validation failed) for subsequent redisplay.
- Struts provides numerous, custom JSP tags that are simple to use but are powerful in the sense that they hide information. Page Designer does not need to know much about form beans, for example, beyond the bean names and the names of each field in a given bean.

The Struts contribution to the controller component of the MVC design pattern is as follows:

- The Struts action servlet handles run-time events in accordance with a set of rules that are provided at deployment time. Those rules are contained in a Struts configuration file and specify how the servlet responds to every outcome received from the business logic. Changes to the flow of control require changes only to the configuration file.
- Struts also provides the Java class `org.apache.struts.action.Action`, which a Java developer subclasses to create an "action class". At run time, the action servlet is said to "execute actions," which means that the servlet invokes the `execute` method of each of the instantiated action classes. The object returned from the `execute` method directs the action servlet as to what action or JSP file to access next.

We recommend that you promote reuse by invoking business logic from the action class rather than including business logic in that class.

Struts does not contribute directly to model development. The model comprises the persistent data (typically stored in a database) and the business logic that accesses that data. The model is not included in a Web diagram. However, the Struts actions and configuration file provide an elegant way to control the circumstances under which the model components are invoked.

Struts Flow of Control:

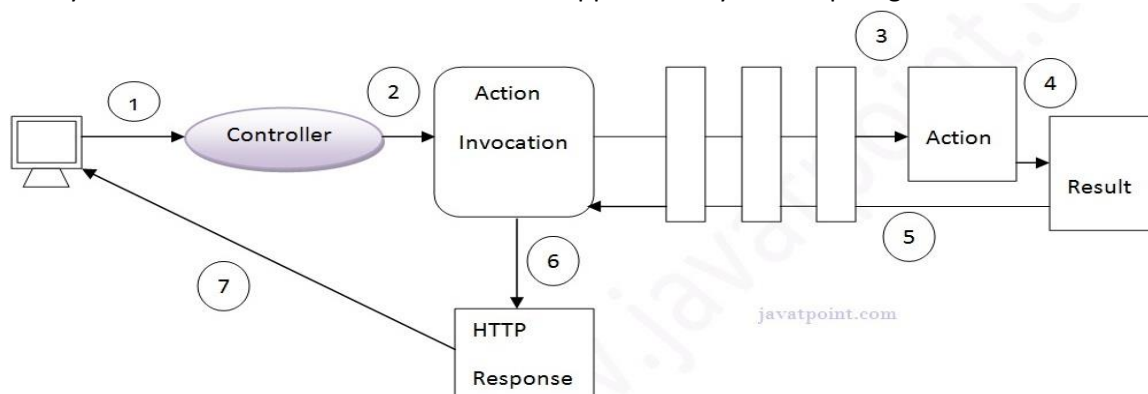
The **architecture and flow of struts 2 application**, is combined with many components such as Controller, ActionProxy, ActionMapper, Configuration Manager, ActionInvocation, Inceptor, Action, Result etc.

Here, we are going to understand the struts flow by 2 ways:

1. struts 2 basic flow
2. struts 2 standard architecture and flow provided by apache struts

Struts 2 basic flow

Let's try to understand the basic flow of struts 2 application by this simple figure:



1. User sends a request for the action
2. Controller invokes the ActionInvocationss

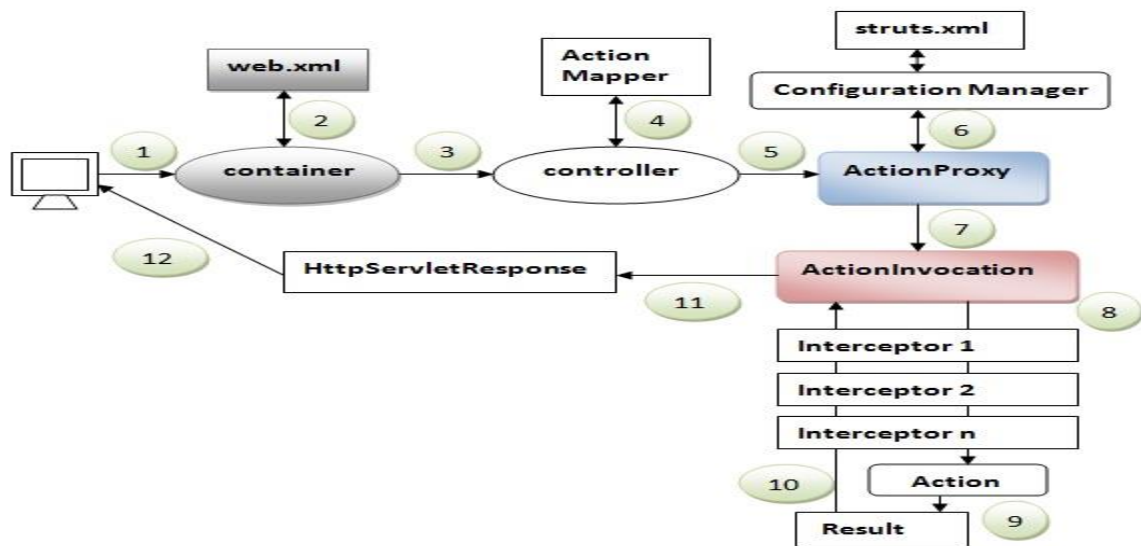
Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

3. ActionInvocation invokes each interceptors and action
4. A result is generated
5. The result is sent back to the ActionInvocation
6. A HttpServletResponse is generated
7. Response is sent to the user

Struts 2 standard flow (Struts 2 architecture)

Let's try to understand the standard architecture of struts 2 application by this simple figure:



1. User sends a request for the action
2. Container maps the request in the web.xml file and gets the class name of controller.
3. Container invokes the controller (StrutsPrepareAndExecuteFilter or FilterDispatcher). Since struts2.1, it is StrutsPrepareAndExecuteFilter. Before 2.1 it was FilterDispatcher.
4. Controller gets the information for the action from the ActionMapper
5. Controller invokes the ActionProxy
6. ActionProxy gets the information of action and interceptor stack from the configuration manager which gets the information from the struts.xml file.
7. ActionProxy forwards the request to the ActionInvocation
8. ActionInvocation invokes each interceptors and action
9. A result is generated
10. The result is sent back to the ActionInvocation
11. A HttpServletResponse is generated
12. Response is sent to the user

Processing Requests with Action Objects

When a client's request matches the action's name, the framework uses the mapping from struts.xml file to process the request. The mapping to an action is usually generated by a Struts Tag. The action tag (within the struts root node of struts.xml file) specifies the action by name and the framework renders the default extension and other required stuff.

The default entry method to the handler class is defined by the Action interface.

Struts Action interface

All actions may implement this interface, which exposes the execute() method. You are free to create

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

POJOs that maintains the same contract defined by this interface without actually implementing the interface.

```
package com.opensymphony.xwork2;
public interface Action {
    //The action execution was a failure.
    public final static String ERROR;
    //The action execution require more input in order to succeed
    public final static String INPUT;
    //The action could not execute, since the user most was not logged in
    public final static String LOGIN;
    // The action execution was successful but do not show a view.
    public final static String NONE;
    //The action execution was successful.
    public final static String SUCCESS;
    //the logic of the action is implemented
    public String execute() throws Exception;
}
```

Implementing the Action interface is optional. If Action is not implemented, the framework will use reflection to look for an execute method. If there is no execute method and no other method is specified in the configuration, the framework will throw an exception.

The Struts2 Action have introduced a simpler implementation approach of the action classes as POJO(Plain Old Java Objects). The most basic usage of an action, is to perform work with a single result always being returned. It looks like:

```
public class NewAction {
    public String execute() throws Exception {
        // do the work
        return "success";
    }
}
```

Note that here the action class doesn't extend any other class or interface. The method invoked in the processing of an action is the "execute" method. The "execute" method has no parameter and it returns a String object. However with struts 2 actions you can get different return types other than the string objects, by using helper interfaces available (Helper interface provides the common results as constants like "success", "none", "error", "input" and "login").

The Action Class usually acts as a Model and executes a particular business logic depending on the Request object and the Input Parameters. In earlier versions of Struts (before Struts 2.0), an Action class is supposed to extend the org.apache.struts.Action class and has to override the Action.execute() method which takes four parameters.

Basically, actions are meant to process various objects like HttpServletRequest, HttpServletResponse. But here, our action class is not processing any object parameter. Here one important issue arises - how do you get access to the objects that you need to work with? The answer lies in the "inversion of control" or "dependency injection" pattern.

To provide a loosely coupled system, Struts2 uses a technique called dependency injection, or inversion of control. Dependency injection can be implemented by constructor injection, interface injection and setter injection. Struts uses setter injection. This means that to have objects available to the action, we need only to provide a setter method. There are also objects such as the HttpServletRequest that can be obtained by asking the ActionContext or implementing ServletRequestAware. Implementing

Advance Java Programming

ServletRequestAware is preferred. For each of the non-business objects there is a corresponding interface (known as an ?aware? interface) that the action is required to implement.

To understand the inversion of control better, let's look at an example where the processing of the action requires access to the current requests HttpServletRequest object.

The dependency injection mechanism used in this example is interface injection. As the name implies, with interface injection there is an interface that needs to be implemented. This interface contains setter methods, which in turn are used to provide data to the action. In our example we are using the ServletRequestAware interface, here it is:

```
public interface ServletRequestAware {  
    public void setServletRequest(HttpServletRequest request);  
}
```

When we implement this interface, our simple action gets modified a bit but now we have a HttpServletRequest object to use.

```
public class NewAction implements addDependency {  
    private HttpServletRequest request;  
    public void setServletRequest(HttpServletRequest request) {  
        this.request = request;  
    }  
    public String execute() throws Exception {  
        // do the work using the request  
        return "SUCCESS";  
    }  
}
```

In Struts, an action instance is created for each request. It's not shared and it's discarded after the request has been completed.

Action Mappings

The action mapping can specify a set of result types, a set of exception handlers, and an interceptor stack. But, only the name attribute is required. The other attributes can also be provided at package scope. Specified through Struts.xml file.

The configuration for this action looks like this:

```
<action name="new" class="NewAction"  
>  
<result>view.jsp</result>  
</action>
```

The ?name? attribute provides the URL information to execute the action as ?/new.action?. The extension ?action? is configured in the ?struts.properties? configuration file. The ?class? attribute provides the full package and class name of the action to be executed.

Struts processes an action class as a POJO and enables to return different results depending on the outcome of the logic. To get different results, the class needs to get modified as:

```
class NewAction {  
    public void String execute() throws Exception {  
        if ( its-ok() ) {  
            return "login";  
        }  
    }  
}
```

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
}  
else {  
return "none";  
}}}
```

Here the class provides two different results, we have to configure the action tags of struts.xml file for each case. Modify the configuration as :

```
<action name="new" class="NewAction" >  
<result>login.jsp</result>  
<result name="none">none.jsp</result>  
</action>
```

Handling Request Parameters with FormBeans

A form bean is a type of Java™ bean. A form bean is an instance of a subclass of an ActionForm class, which stores HTML form data from a submitted client request or that can store input data from a Struts action link that a user clicked. An HTML form comprises fields in which the user can enter information. A form-bean mapping is an entry in a Struts configuration file that maps a form bean to an action.

When a browser submits an HTML form, the Struts action servlet does as follows:

1. Looks at the field names from the HTML form
2. Matches them to the properties' names in the form bean
3. Automatically calls the set methods of these variables to put the values retrieved from the HTML form

In addition, if you implement a validate method and set the validate flag in the corresponding action mapping entry in the Struts configuration file, the action servlet invokes the validate method to validate that the data that the servlet receives is of the appropriate type.

Prepopulating and Redisplaying Input Forms

Using the Struts html: tags to build HTML forms that have two important characteristics:

- They can be prepopulated based on the values in a JavaBean.
 - That is, the initial values of the form elements can be taken from a Java object.
- They can be redisplayed when they are submitted with incomplete or incorrect values.
 - Specifically, when they are redisplayed, they can maintain the values that the end user already entered.

Add an input attribute to the action element in struts-config.xml.

- This attribute tells the system to associate a form-bean of the type designated by name (which should match a name in the form-bean element) with the input form.
 - Use the appropriate scope in the action declaration.
- Use scope="request" if you want to prepopulate a form only (i.e., display initial values).
- Use scope="session" if also you want to redisplay the form with previous values intact (i.e., redisplay an incomplete form without making the user reenter values they already entered).
- Default scope is session (surprisingly), but always list it explicitly.

Use html:form to declare the input form in the initial JSP page.

- The address of this input page must match the input attribute of the action element from the previous step.
 - Using the Struts html:form element instead of the standard HTML FORM element yields four results:
 - A bean is associated with the form.
 - A bean of the type specified by form-bean is automatically used
 - The Web application prefix is prepended automatically.
 - You say <html:form action="/actions/..."> to get

Advance Java Programming

<FORM ACTION="/webAppPrefix/actions/..." ...>.

– The .do suffix is appended automatically.

• You say <html:form action="/actions/blah"> to get

<FORM ACTION="/webAppPrefix/actions/blah.do" ...>.

– POST, not GET, is the default METHOD.

• You say <html:form action="/actions/blah"> to get

<FORM ACTION="/webAppPrefix/actions/blah.do" METHOD="POST">.

Use html:text and similar elements to declare the input fields of the form.

– The NAME of each input field is taken from the bean property name, and the VALUE is taken from the bean property value. For example, using

<html:text property="firstName"/>

is equivalent to first declaring a bean of the appropriate type, then doing

<INPUT TYPE="TEXT" NAME="firstName" VALUE="<%= theBean.getFirstName() %>">.

– Not only does this provide initial values for your formfields, but it also makes it easier for you to be sure that the field names match the bean property names. Since, in the execute method of the Action object, the form-bean is filled in by matching up request parameter names with bean property names, it is critical that the names stay in synch.

Using Properties Files

The struts.properties file

This configuration file provides a mechanism to change the default behavior of the framework. Actually all of the properties contained within the struts.properties configuration file can also be configured in the web.xml using the init-param, as well using the constant tag in the struts.xml configuration file. But if you like to keep the things separate and more struts specific then you can create this file under the folder WEB-INF/classes.

The values configured in this file will override the default values configured in default.properties which is contained in the struts2-core-x.y.z.jar distribution. There are a couple of properties that you might consider changing using struts.properties file:

When set to true, Struts will act much more friendly for developers

struts.devMode = true

Enables reloading of internationalization files

struts.i18n.reload = true

Enables reloading of XML configuration files

struts.configuration.xml.reload = true

Sets the port that the server is run on

struts.url.http.port = 8080

Steps to Create Struts Hello World application

Step 1: Download the Struts 2.3.15.1 and extract the zip file.

Step 2: You should download and install latest version of Eclipse IDE on your computer. Also configure the Tomcat 7 on Eclipse IDE so that you can run the examples from the Eclipse IDE itself.

Step 3: Open the Eclipse IDE and create a dynamic web project as shown below:

Name the project "**Struts2HelloWorld**" as shown below:

Eclipse IDE will create the project and it will display the project in the Project explorer window as shown below:

Step 4: Create a new package "net.bcaty". Now create a new Java file "**HelloWorld.java**" under the package "net.bcaty" and add the following code:

```
package net.bcaty;
```

```
import com.opensymphony.xwork2.ActionSupport;
```

```
public class HelloWorld extends ActionSupport {
```

Smt J.J.Kundalia Commerce College Department Of Computer Science
Advance Java Programming

```
public String execute() throws Exception {
    setMessage(getText(MESSAGE));
    return SUCCESS;
}
/**
 * Provide default value for Message property.
 */
public static final String MESSAGE = "HelloWorld.message";
/**
 * Field for Message property.
 */
private String message;
/**
 * Return Message property.
 *
 * @return Message property
 */
public String getMessage() {
    return message;
}
/**
 * Set Message property.
 *
 * @param message Text to display on HelloWorld page.
 */
public void setMessage(String message) {
    this.message = message;
}
}
```

The "**HelloWorld.java**" is our Action class which contains a variable message whose value is picked up from a package.properties and package_es.properties files. You will get the files from the source code of this tutorial. You should copy and add the package.properties and package_es.properties files from our download source code.

Step 5: Create index.jsp file in the "WebContent" directory of the project. Add the following code:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<p>Struts 2 Tutorial</p>
<p><a href="/Struts2HelloWorld/bcaty/HelloWorld.action">Test Struts 2
    Hello World example!!</a>
```

Smt J.J.Kundalia Commerce College Department Of Computer Science

Advance Java Programming

```
</body>
</html>
```

In the above code we have the link to execute the Hello World example. Link url is

/Struts2HelloWorld/roseindia/HelloWorld.action.

Step 6: Create a new folder names "classes" under WEB-INF folder of the application and create a new xml file "struts.xml" and add following code into it:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
"http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
<constant name="struts.enable.DynamicMethodInvocation" value="false" />
<constant name="struts.devMode" value="true" />
<package name="default" namespace="/bcaty" extends="struts-default">
<default-action-ref name="index" />
<action name="HelloWorld" class="net.bcaty.HelloWorld">
<result>/example/HelloWorld.jsp</result>
</action>
</package>
</struts>
```

In the above code we are declaring our action class "net.roseindia.HelloWorld". The action name given is "HelloWorld" and we will call this action from browser by typing

<http://localhost:8080/Struts2HelloWorld/roseindia/HelloWorld.action> in the browser. Struts 2 framework determines the action to be executed through the action name.

```
<action name="HelloWorld" class="net.roseindia.HelloWorld">
  <result>/example/HelloWorld.jsp</result>
</action>
```

Once the action is successfully executed the **/example/HelloWorld.jsp** will be used to create the view.

Step 7: Create a new directory example under WebContent folder of the application and then create a new jsp page "**HelloWorld.jsp**" and add following code:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title><s:text name="HelloWorld.message"/></title>
</head>
<body>
<h2><s:property value="message"/></h2>
<a href="/Struts2HelloWorld/">Back to Index page</a>
</body>
</html>
```

Here in the above code **<s:property value="message"/>** tag is used to print the value of message variable in the action class (HelloWorld java).

Advance Java Programming

Step 8: Finally add the required jar files from the **struts-2.3.15.1-all.zip** zip file. You have to extract the struts-2.3.15.1-all.zip file and copy the following jar files from the **lib** directory into **WebContent/WEB-INF/lib** directory of the Eclipse project:

- asm-3.3.jar
- asm-commons-3.3.jar
- asm-tree-3.3.jar
- commons-fileupload-1.3.jar
- commons-io-2.0.1.jar
- commons-lang3-3.1.jar
- commons-logging-1.1.3.jar
- freemarker-2.3.19.jar
- javassist-3.11.0.GA.jar
- log4j-1.2.17.jar
- ognl-3.0.6.jar
- struts2-core-2.3.15.1.jar
- xwork-core-2.3.15.1.jar

Step 9: We are done with the work and to test the project, right click on the "**Struts2HelloWorld**" in the project explorer window and then select **Run as** and then "**Run on Server**" as shown below:

Select the "**Tomcat v7.0 Server at localhost**" and then click **Finish** button. Eclipse will compile and then deploy the application on Tomcat 7. After successful startup of the application Eclipse will open the url <http://localhost:8080/Struts2HelloWorld/> in the browser as shown below:

To run the example click on the link "**Test Struts 2 Hello World example !!**". Browser should display the message as shown below:

You have successfully developed Hello World application in Struts 2