**J2EE-19**

**Q-1[A]ONE MARK:**

1]which logic resides at the middle tier?

a. Business logic

2]which API provides server side software component level functionality in j2ee application?

a. Enterprise Java Beans (EJB)

3]which package needs to be imported for implementing JDBC API in a JDBC program?

a. The javax

4]which interface is used to call stored procedure in JDBC?

A. PreparedStatement interface

**Q-1[B]TWO MARK:**

1]define:java mail

a. The JavaMail is an API that is used to compose, write and read electronic messages (emails).

The JavaMail API provides protocol-independent and plateform-independent framework for sending and receiving mails.

The javax.mail and javax.mail.activation packages contains the core classes of JavaMail API.

The JavaMail facility can be applied to many events. It can be used at the time of registering the user (sending notification such as thanks for your interest to my site), forgot password (sending password to the users email id), sending notifications for important updates etc. So there can be various usage of java mail api.

2]what is web container?

a. Web Container is a java application that controls servlet. Servlet does not have a main() method, So they require a container to load them.

Container is a place where servlet gets deployed.

When a client sends a request to web server that contains a servlet, the server sends that request to the container rather than to servlet directly. Container then finds out the requested servlet and pass the Http Request and response to servlet and loads the servlet methods i.e. doGet() or do Post().

An example of a web container is Tomcat.

**Q-1[C]THREE MARK:**

1]what is j2ee?explain j2ee platform using j2ee API

a. J2EE is a collection of APIs which could be used to build such large-scale systems. J2EE is aimed to be a standard for building and deploying enterprise applications, held together by the specifications of the APIs that it defines and the services that J2EE provides.

The purpose of the J2EE platform is to develop multi-tier, web-based applications using a series of protocols and application programming interfaces (APIs). J2EE provides the enterprise level with a Java platform that is fast, safe and secure, improving on the J2SE. (Java 2 Platform, Standard Edition).

2]list and explain any two types of JDBC driers

a. JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

**Type-1 driver**

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

* As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
* The ODBC bridge driver is needed to be installed in individual client machines.
* Type-1 driver isn’t written in java, that’s why it isn’t a portable driver.
* This driver software is built-in with JDK so no need to install separately.
* It is a database independent driver.

**Type-2 driver**

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that’s why data transfer is much more secure as compared to type-1 driver.

* Driver needs to be installed separately in individual client machines
* The Vendor client library needs to be installed on client machine.
* Type-2 driver isn’t written in java, that’s why it isn’t a portable driver
* It is a database dependent driver.

**Type-3 driver**

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

* Type-3 drivers are fully written in Java, hence they are portable drivers.
* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
* Network support is required on client machine.
* Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
* Switch facility to switch over from one database to another database.

**Type-4 driver**

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

* Does not require any native library and Middleware server, so no client-side or server-side installation.
* It is fully written in Java language, hence they are portable drivers.

**Q-1[D]FIVE MARK:**

1]what are enterprice architecture styles?explain N-tier architecture in detail using appropriate diagram

a.

2]explain the JDBC-mysql connectivity stepwise in detail with example

a. To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver.

Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.

Username: The default username for the mysql database is root.

Password: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;
2. use sonoo;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password both.

1. **import** java.sql.\*;
2. **class** MysqlCon{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/sonoo","root","root");
8. //here sonoo is database name, root is username and password
9. Statement stmt=con.createStatement();
10. ResultSet rs=stmt.executeQuery("select \* from emp");
11. **while**(rs.next())
12. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
13. con.close();
14. }**catch**(Exception e){ System.out.println(e);}
15. }
16. }

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

|  |
| --- |
| Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here. |

2) Set classpath:

|  |
| --- |
| There are two ways to set the classpath:   * temporary * permanent |

How to set the temporary classpath

|  |
| --- |
| open command prompt and write: |

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

**Q-2[A]ONE MARK:**

1]which remote object act as a client local representative or proxy for the remote object in RMI?

a. A stub for a remote object acts as a client's local representative or proxy for the remote object

2]RMI STAND FOR- Remote Method Invocation (RMI)

3]which package needs to be imported for servlet?

a. javax. servlet.

4]which file known as deployment descriptor

a. web.xml

**Q-2[B]TWO MARK:**

1]explain stub and skeleton

a. stub

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

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

skeleton

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

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

2]what is servlet?

a. Servlet can be described in many ways, depending on the context.

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

**Q-2[C]THREE MARK:**

1]what is RMI?explain RMI architecture

a. The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java.

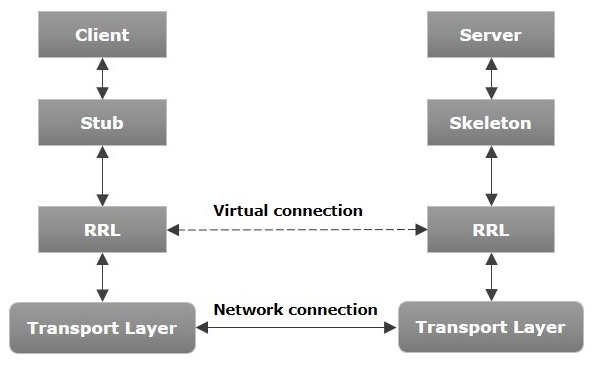
The RMI provides remote communication between the applications using two objects stub and skeleton.

\*Architecture of an RMI Application\*

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

* Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
* The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



Let us now discuss the components of this architecture.

* Transport Layer − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
* Stub − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
* Skeleton − This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.
* RRL**(**RemoteReferenceLayer**)** − It is the layer which manages the references made by the client to the remote object.

2]explain session mechanism in servlet

a.

**Q-2[D]FIVE MARK:**

1]servlet life cycle

a. The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(ServletConfig config) throws ServletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

2]program

**Q-3[A]ONE MARK:**

1]which tag allow to write java code into jsp file?

a. scriptlet tag

2]which tag is used to start and end jsp in a jsp page?

a. scriplet tag

3]which tag is used for plugin mechanism in jsp page?

a. action

4]what are special classes that are serializable having a zero argument constructor and allow access to properties using getter and setter method and can be reused in j2ee application?

a. JavaBeans

**Q-3[B]TWO MARK:**

1]what are including and forwarding techniques of jsp pages?

a.

2]define:jsp implicit objects

a.

there are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

|  |  |
| --- | --- |
| **Object** | **Type** |
| out | JspWriter |
| request | HttpServletRequest |
| response | HttpServletResponse |
| config | ServletConfig |
| application | ServletContext |
| session | HttpSession |
| pageContext | PageContext |
| page | Object |
| exception | Throwable |

**Q-3[C]THREE MARK:**

1]jsp action elements

a. There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

|  |  |
| --- | --- |
| **S.No.** | **Syntax & Purpose** |
| 1 | **jsp:include**  Includes a file at the time the page is requested. |
| 2 | **jsp:useBean**  Finds or instantiates a JavaBean. |
| 3 | **jsp:setProperty**  Sets the property of a JavaBean. |
| 4 | **jsp:getProperty**  Inserts the property of a JavaBean into the output. |
| 5 | **jsp:forward**  Forwards the requester to a new page. |
| 6 | **jsp:plugin**  Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin. |
| 7 | **jsp:element**  Defines XML elements dynamically. |
| 8 | **jsp:attribute**  Defines dynamically-defined XML element's attribute. |
| 9 | **jsp:body**  Defines dynamically-defined XML element's body. |
| 10 | **jsp:text**  Used to write template text in JSP pages and documents. |

2]noteLjavabean properties and method

a. A JavaBean is a Java class that should follow the following conventions:

* It should have a no-arg constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.
* 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 the 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 −

|  |  |
| --- | --- |
| S.No. | Method & Description |
| 1 | getPropertyName()  For example, if property name is *firstName*, your method name would be getFirstName() to read that property. This method is called accessor. |
| 2 | setPropertyName()  For example, if property name is *firstName*, your method name would be setFirstName() to write that property. This method is called mutator. |

**Q-3[D]FIVE MARK:**

1]life cycle of jsp page with method

a. >Compilation of JSP page :-

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

>Classloading :-

Servlet class which has been loaded from JSP source is now loaded into container.

>Instantiation :-

Here instance of the class is generated. The container manages one or more instance by providing response to requests.

>Initialization :-

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

>Request processing :-

\_jspService() method is used to serve the raised requests by JSP.It takes request and response object as parameters.This method cannot be overridden.

>JSP Cleanup :-

In order to remove the JSP from use by the container or to destroy method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

|  |
| --- |
| Method Detail |

jspInit

public void jspInit()

The jspInit() method is invoked when the JSP page is initialized. It is the responsibility of the JSP implementation (and of the class mentioned by the extends attribute, if present) that at this point invocations to the getServletConfig() method will return the desired value. A JSP page can override this method by including a definition for it in a declaration element. A JSP page should redefine the init() method from Servlet.

jspDestroy

public void jspDestroy()

The jspDestroy() method is invoked when the JSP page is about to be destroyed. A JSP page can override this method by including a definition for it in a declaration element. A JSP page should redefine the destroy() method from Servlet.

2]explain directive elements of jsp in detail

a. Directives are elements that relay messages to the JSP container and affect how it compiles the JSP page.

There are three types of directives:

* page directive
* include directive
* taglib directive

|  |  |
| --- | --- |
| S.No. | Directive & Description |
| 1 | <%@ page ... %>  Defines page-dependent attributes, such as scripting language, error page, and buffering requirements. |
| 2 | <%@ include ... %>  Includes a file during the translation phase. |
| 3 | <%@ taglib ... %>  Declares a tag library, containing custom actions, used in the page |

**JSP - The page Directive**

The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive −

<%@ page attribute = "value" %>

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.page attribute = "value" />

Attributes

Following table lists out the attributes associated with the page directive −

|  |  |
| --- | --- |
| S.No. | Attribute & Purpose |
| 1 | buffer  Specifies a buffering model for the output stream. |
| 2 | autoFlush  Controls the behavior of the servlet output buffer. |
| 3 | contentType  Defines the character encoding scheme. |
| 4 | errorPage  Defines the URL of another JSP that reports on Java unchecked runtime exceptions. |
| 5 | isErrorPage  Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute. |
| 6 | extends  Specifies a superclass that the generated servlet must extend. |
| 7 | import  Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes. |
| 8 | info  Defines a string that can be accessed with the servlet's getServletInfo() method. |
| 9 | isThreadSafe  Defines the threading model for the generated servlet. |
| 10 | language  Defines the programming language used in the JSP page. |
| 11 | session  Specifies whether or not the JSP page participates in HTTP sessions |
| 12 | isELIgnored  Specifies whether or not the EL expression within the JSP page will be ignored. |
| 13 | isScriptingEnabled  Determines if the scripting elements are allowed for use. |

**The include Directive**

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include*** directives anywhere in your JSP page.

The general usage form of this directive is as follows −

<%@ include file = "relative url" >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.include file = "relative url" />

For more details related to include directive, check the [Include Directive](https://www.tutorialspoint.com/jsp/include_directive.htm).

**The taglib Directive**

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below −

<%@ taglib uri="uri" prefix = "prefixOfTag" >

Here, the uri attribute value resolves to a location the container understands and the prefix attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />

**Q-4[A]ONE MARK:**

1]full form EJB- Enterprise Java Beans

2]in MVC…...defines java beans classes that internally define the system state and actions to change that state?

a.

3]ORM stand for- Object-relational mapping

4]which provides mapping information in hibernate?

a.

**Q-4[B]TWO MARK:**

1]define term:MVC

A. MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

* **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate

2]full form BMP & CMP

a.  container-managed persistence (**CMP**) and bean-managed persistence (**BMP**).

**Q-4[C]THREE MARK:**

1]types of EJB

a. . EJB is an acronym for enterprise java bean. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.

There are 3 types of enterprise bean in java.

* Session Bean

Session bean contains business logic that can be invoked by local, remote or webservice client.

* Message Driven Bean

Like Session Bean, it contains the business logic but it is invoked by passing message.

* Entity Bean

It encapsulates the state that can be persisted in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API).

2]explain hibernate configuration file

a. As Hibernate can operate in different environments, it requires a wide range of configuration parameters. These configurations contain the mapping information that provides different functionalities to Java classes. Generally, we provide database related mappings in the configuration file. Hibernate facilitates to provide the configurations either in an XML file (like hibernate.cfg.xml) or properties file (like hibernate.properties).

An instance of Configuration class allows specifying properties and mappings to applications. This class also builds an immutable **SessionFactory**.

We can acquire the Configuration class instance by instantiating it directly and specifying mappings in the configuration file. Use the addResource() method, if the mapping files are present in the classpath.

1. Configuration cfg = new Configuration()
2. .addResource("employee.hbm.xml")

Hibernate Configuration Properties

Property Description

hibernate.dialect It represents the type of database used in hibernate to generate SQL statements for a particular relational database.

hibernate.show\_sql It is used to display the executed SQL statements to console.

hibernate.format\_sql It is used to print the SQL in the log and console.

hibernate.default\_catalog It qualifies unqualified table names with the given catalog in generated SQL.

hibernate.default\_schema It qualifies unqualified table names with the given schema in generated SQL.

hibernate.session\_factory\_name The SessionFactory interface automatically bound to this name in JNDI after it has been created.

hibernate.default\_entity\_mode It sets a default mode for entity representation for all sessions opened from this SessionFactory

hibernate.order\_updates It orders SQL updates on the basis of the updated primary key.

hibernate.use\_identifier\_rollback If enabled, the generated identifier properties will be reset to default values when objects are deleted.

hibernate.generate\_statistics If enabled, the Hibernate will collect statistics useful for performance tuning.

hibernate.use\_sql\_comments If enabled, the Hibernate generate comments inside the SQL. It is used to make debugging easier.

**Q-4[D]FIVE MARK:**

1]explain benefits of EJB

a. Following are the important benefits of EJB −

* Simplified development of large-scale enterprise level application.
* Application Server/EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling, and so on. Developer has to focus only on business logic of the application.
* EJB container manages life cycle of EJB instances, thus developer needs not to worry about when to create/delete EJB objects.

2]what is annotation in hibernate?explain the hibernate inheritance in detail.

a. The hibernate application can be created with annotation. There are many annotations that can be used to create hibernate application such as @Entity, @Id, @Table etc.

Hibernate Annotations are based on the JPA 2 specification and supports all the features.

All the JPA annotations are defined in the **javax.persistence** package. Hibernate EntityManager implements the interfaces and life cycle defined by the JPA specification.

The core advantage of using hibernate annotation is that you don't need to create mapping (hbm) file. Here, hibernate annotations are used to provide the meta data.

We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

1. Table Per Hierarchy
2. Table Per Concrete class
3. Table Per Subclass

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 .

[Table Per Hierarchy using xml file](https://www.javatpoint.com/hibernate-table-per-hierarchy-example-using-xml-file)  
[Table Per Hierarchy using Annotation](https://www.javatpoint.com/hibernate-table-per-hierarchy-using-annotation-tutorial-example)

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.

[Table Per Concrete class using xml file](https://www.javatpoint.com/table-per-concrete-class)  
[Table Per Concrete class using Annotation](https://www.javatpoint.com/hibernate-table-per-concrete-class-using-annotation-tutorial-example)

Table Per Subclass

In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.

[Table Per Subclass using xml file](https://www.javatpoint.com/table-per-subclass)  
[Table Per Subclass using Annotation](https://www.javatpoint.com/hibernate-table-per-subclass-using-annotation-tutorial-example)

**Q-5[A]ONE MARK:**

1]which is a lightweight ,loosely coupled integrated framework for developing enterprice level java applications?

a. J2EE (Java 2 Enterprise Edition)

2]ioc stand for- inversion of control (IoC)

3]AOP stand for- aspect-oriented programming (AOP)

4]strut based application …in design pattern of an application

a. model-view-controller (MVC) design pattern.

**Q-5[B]TWO MARK:**

1]definition :spring context

a. Spring contexts are also called Spring IoC containers, which are responsible for instantiating, configuring, and assembling beans by reading configuration metadata from XML, Java annotations, and/or Java code in the configuration files.

2]what is struts?

a. Struts is used to create a web applications based on servlet and JSP. Struts depend on the MVC (Model View Controller) framework. Struts application is a genuine web application. Struts are thoroughly useful in building J2EE (Java 2 Platform, Enterprise Edition) applications because struts takes advantage of J2EE design patterns. Struts follows these J2EE design patterns including MVC.

**Q-5[C]THREE MARK:**

1]write note:AOP in spring

a. Aspect Oriented Programming (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by cross-cutting concerns.

A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

Why use AOP?

It provides the pluggable way to dynamically add the additional concern before, after or around the actual logic.

Where use AOP?

AOP is mostly used in following cases:

* to provide declarative enterprise services such as declarative transaction management.
* It allows users to implement custom aspects.

2]write note:properties file in a struts application

## a. The Struts.properties File

This configuration file provides a mechanism to change the default behavior of the framework. Actually, all 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 the 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

Here any line starting with hash (#) will be assumed as a comment and it will be ignored by Struts 2.

**Q-5[D]FIVE MARK:**

1]spring architecture

a. The Spring Framework provides about 20 modules which can be used based on an application requirement.

Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows −

* The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
* The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
* The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
* The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows −

* The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
* The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
* The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
* The Java Messaging Service **JMS** module contains features for producing and consuming messages.
* The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows −

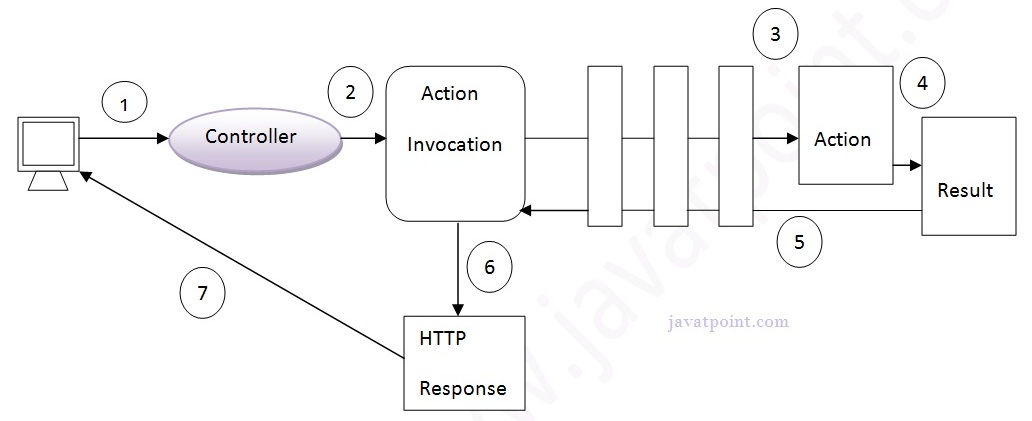
* The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
* The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
* The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
* The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Miscellaneous

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules the details of which are as follows −

* The **AOP** module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
* The **Aspects** module provides integration with AspectJ, which is again a powerful and mature AOP framework.
* The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.
* The **Messaging** module provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
* The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.

2]explain struts flow diagram

a. 

1. User sends a request for the action
2. Controller invokes the ActionInvocation
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