Page No.
Date

⊛ Chapter: ⑥ : Hibernate ⊛

Q.① What is Hibernate? List Advantages of Hibernate over JDBC.

- Hibernate is used to convert object data in Java to relational databases tables.
- It is an open source Object-Relational Mapping (ORM) for Java.
- Hibernate is responsible for making data persistent by storing it in a database.

⊛ Advantages of Hibernate :-

- Hibernate is flexible and powerful ORM to map Java classes to databases tables.
- Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
- Hibernate automatically generates the queries.
- It makes an application portable to all SQL database.

- Handles all create-read-update-delete (CRUD) operations using simple API; no SQL

- Hibernate itself take care of this mapping using XML files so developer does not need to write code for this.

- Hibernate provides a powerful query language Hibernate Query Language - HQL (independent from type of databases).

- Hibernate supports Inheritance, Associations, Collections.

- Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many, Many-To-one.

- Hibernate provided Dialect classes, so we no need to write SQL queries in hibernate, instead we use the methods provided by that API.

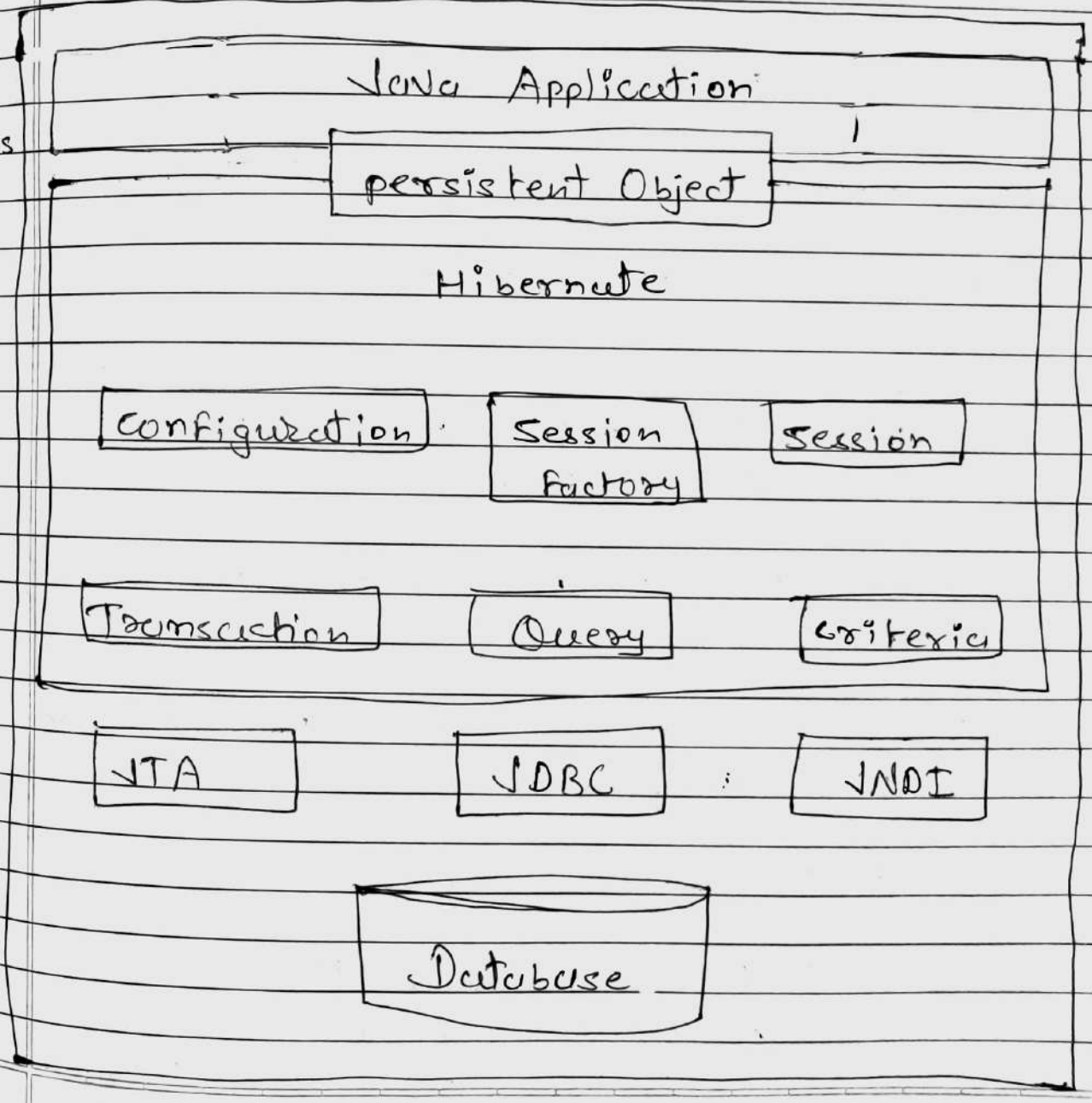
	JDBC	Hibernate
(*)	(*) JDBC maps Java classes to database tables (and from Java data types to SQL data types)	Hibernate automatically generates the queries.

(8) JDBC	Hibernate
(2) With JDBC, developer has to write code to map an object model's data to a relational data model.	Hibernate is flexible and powerful ORM to map Java classes to databases tables.
(3) With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java. So with JDBC, mapping between Java objects and database table is done manually.	Hibernate reduces ~~rites~~ lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance set.
(4) Require JDBC Driver for different types of databases.	Makes an application portable to all SQL databases.
(5) Handles all 'create-read-update-delete' (CRUD) operations using SQL Queries.	Handles all (CRUD) operations using simple API; no SQL
(6) Working with both Object-Oriented Software & Relational Database is complicated task with JDBC.	Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

(5) JDBC supports only native Structured Query Language (SQL)	Hibernate provides a powerful Q.lang. Hibernate-Query-Language (HQL).

Q. (2) Architecture of Hibernate.

(1) Diagram :-



(2) Architecture of Hibernate (2)

- For creating the first hibernate application, we must know the objects / elements of Hibernate architecture:

- They are as follows:

(1) Configuration
(2) Session Factory
(3) Session
(4) Transaction factor
(5) Query
(6) Criteria

[1] Configuration Object :-

- The Configuration object is the first Hibernate object you create in any Hibernate application.

- It is usually created only once during application initialization.

- The Configuration object provides two keys components:

(1) Database Connection:

- This is handled through one or more configuration files supported by Hibernate.

- These files are hibernate.properties and hibernate.cfg.xml.

(2) class Mapping Setup:

- This component creates the connection between the Java classes & database tables.

[2] SessionFactory Object :-

- The SessionFactor is a thread safe object and used by all the threads of an application.

- Configuration object is used to create a SessionFactory object which in turn configures hibernate for the application.

- You would need one SessionFactory object per databases using a separate configuration file

- So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.

[3] Session Object :-

- A session is used to get a physical connection with a database.

- The session is used to object is lightweight and es designed to be instantiated each time an interaction is needed with the database.

- 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 as needed.

[4] Transaction Object :-

- A Transaction represents a unit of work with the databases & most of the RDBMS supports transaction functionality.

- Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

[5] Query Object :-

- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database & create objects.

- A Query instances is used to bind query parameters, limit the number of result returned by the query, and finally to execute the query.

[6] Criteria Object :-

- Criteria objects are used to create and execute object oriented criteria queries to retrive objects.

Q.③ What is HQL? How does it different from SQL? Lists its advantages.

- The Hibernate ORM framework provides its own query language called Hibernate Query language.

- Hibernate Query Language (HQL) is same as SQL (structured Query language) but it doesn't depends on the table of the databases. Instead of table name, we use class name in HQL.

- Therefore, it is database independent query language.

② SQL	HQL
(1) SQL is based on a relational database model.	HQL is a combination object-oriented programming with relational database
(2) SQL manipulates data stored in tables and modifies its rows & columns.	HQL is concerned about objects and its properties.
(3) SQL is concerned about the relation-ship that exists between two tables.	HQL considers the relation between two objects.

③ Advantages of HQL :-

- provides full support for relation properties.
- Returns results as objects
- support polymorphic queries.
- Easy to learn and use
- supports for advanced features
- provides database independency

Q.④ What is O/R Mapping ? How it is implemented using Hibernate. give an exemple of Hibernate XML mapping file.

- Three most important mapping are as Follows:

of
ing
concepts.

(1) Collections Mappings
(2) Association Mappings
(3) Component Mappings.

[1] Collection Mappings :-

- if an entity or class has collections of values for a particular variable, then we can map these values using any one of the collection interfaces available in java.

- Hibernate can persist instances of java.util.Map, java.util.Set, java.util.Sorted Map, java.util.SortedSet, java.util.List and any array of persistent entities or values.

[2] Association Mappings :-

- The mapping of associations between entity classes and the relationships between tables is the soul of ORM.

- There are the four ways in which the cardinality of the relationship between the objects can be expressed.

- An association mapping can be Unidirectional as well as bidirection

Mapping Type	Description
Many-to-one	Mapping many-to-one
One-to-One	relationship using
One-to-Many	hibernate.
Many-to-Many	" " " .

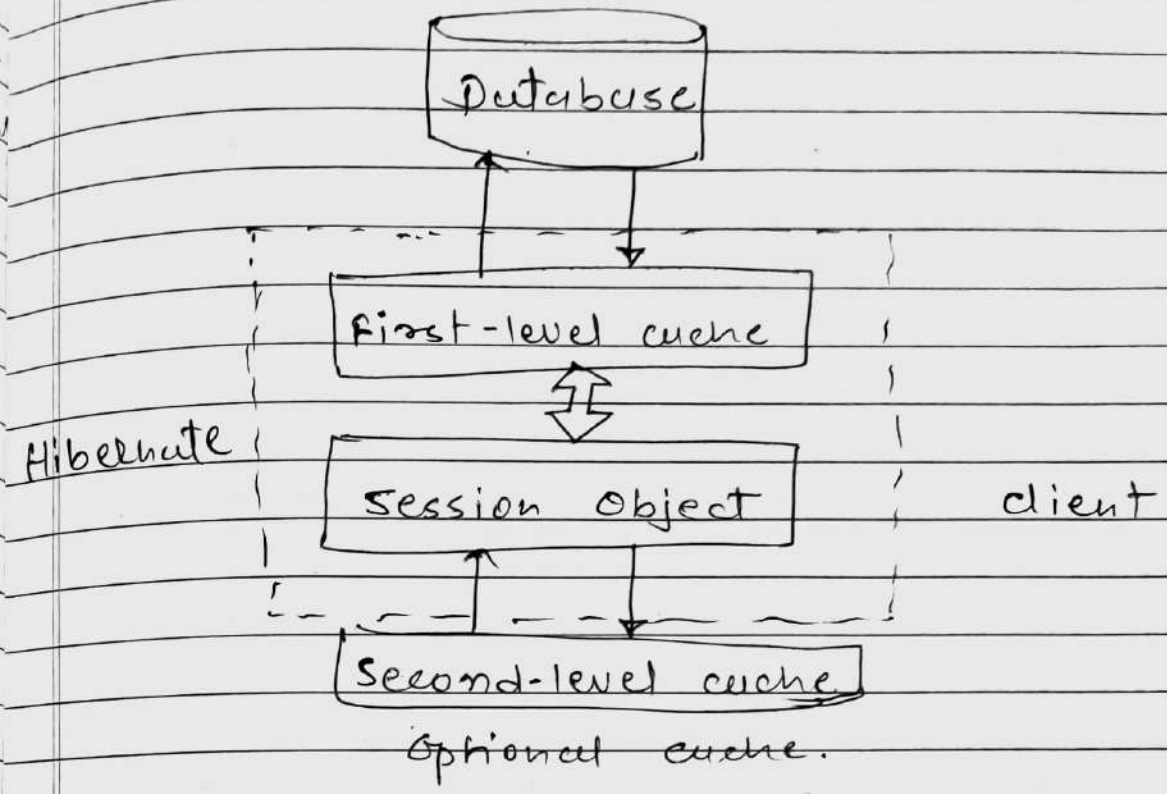
[3] Component Mappings :-

- if the reffered class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the reffere class hence therefore is called as the component class.

- The mapping of collection of components is also possible in a similar way just as the mapping of regular collections with minor configuration differences.

Q.⑧ Hibernate cache Architecture.

- Caching is all about application performance optimization.



⊛ Hibernate cache architecture ⊛

- It is situated between your application and the databases to avoid the numbers of databases hits as many as possible.
- To give a better performance for critical applications.

⊛ First-level cache :-

- The First-level cache is the Session cache.
- The Session object keeps an object under its own control before commiting it to the database.
- If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of

update SQL statements issued.

- If you close the session, all the objects being cached are lost.

(2) Second-level cache :-

- it is responsible for caching objects across sessions.
- second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache.
- Any third party cache can be used with Hibernate.
- while preparing a Hibernate mapping document, we map the Java data types into RDBMS data types. These types are called Hibernate mapping types, which can translate from Java to SQL data types & vice versa.

Q.(5) Hibernate Annotation.

- Hibernate Annotations is the powerful way to provide the metadata for the object and Relational Table mapping.

Annotations :-

(1) @ Entity
(2) @ Table
(3) @ Id
(4) @ GeneratedValue
(5) @ column
(6) @ Transient
(7) @ Temporal
(8) @ Lob
(9) @ OrderBy

(1) @ Entity :

- Used for declaring any POJO class as an entity for a database.
- used to change table details some of the attributes are

(2) @ Table :

- name - Override the table name
- Schema
- catalogue
- enforce unique constraints

(3) @ Id :

- Used for declaring primary key inside our POJO class. [Hibernate automatically generate the values with reference to the internal sequence & we don't need to set the values manually.

(4) @ Generated Value :

⭐ chapter : ① : Java Networking

Q. ① what is server socket? Explain in detail with an example. Discuss the difference betwe the socket and ServerSocket class.

- the ServerSocket class (java.net) can be used to create a server socket.

- this object is used to establish communicatio with the clients.

- A server socket waits for request to come in over the network. it performs some operation based on the request, and then possibly returns a result to the requester

- the actual work of the server socket is performed by an instance of the SocketImp class.

- An Application can change the socket factor that creates the socket implementation to configure itself to create sockets appropiate to the local firewall.

⭐ **Constructor** :→ : Creates a server socket, ServerSocket(int port) bound to the specified port

⭐ **method** :→ : Returns the socket and establis a connection between server & client.

public Socket accept()

Page No.
Date

(*) Syntax → : ServerSocket ss = new ServerSocket (port_no);

(*) Difference Between ServerSocket & Socket class

(*) ServerSocket class :-

- it is placed in server side which sends request to client side socket (Socket) and wait for the response from client

- ServerSocket ss = new ServerSocket (7777);

(*) Socket class :-

- it is placed in client side which sends request to server side socket (serverSocket) and wait for the response from server

- Socket s = new Socket ("localhost", 7777);

Q: (2) What is Datagram Socket? Explain in detail with example.

- Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Page No.
Date

⊛ Datagram Socket :-

- DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

- A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

⊛ Constructor :-

- DatagramSocket() :

- it creates a datagram socket and binds it with the given port Number. on the localhost Machine.

- DatagramSocket (int port):

- it creates ... with the given port Nm.

- DatagramSocket (int port, InetAddress address):

- it creates ... with the specified port nm & host address.

⊛ Datagram Packet :-

- Java DatagramPacket is a message that can be sent or received.

- Additionally, packet delivery is not guaran
- Datagram packets are used to implement a connectionless packet delivery service
- Each message is routed ~~for~~ from one machine to another based solely on information contained within that packet
- Multiple packets sent from one machine another might be ~~routed~~ routed differently, and might ~~be~~ routed arri in any order.

(*) Constructor :-

- DatagramPacket (byte [] barr, int length

  - it creates datagram packet. this constr is used to receive the packets.

- - DatagramPacket (byte [] barr, int leng InetAddress address, int port)

  - it creates a datagram packet. this constructor is used to send packe

(*) Example of Sending DatagramPack by DatagramSocket.

Page No.
Date

② D.sender.java

```
import java.net.*;
public class DSender {
    public static void main (String[] args)
        throws Exception
{
    DatagramSocket ds = new DatagramSocket();
    String str = "message sent by D.socket";
    InetAddress ip = InetAddress.getByName
                    ("127.0.0.1");
    DatagramPacket dp = new DatagramPacket
        (str.getBytes(), str.length(), ip,
                                    3000);
    ds.send(dp);
    ds.close();
}
}
```

③ DReceiver.java

```
import java.net.*;
public class DReceiver {
    public static void main (String args
        throws Exception
{
DatagramSocket ds = new DatagramSocket
                                (3000);
  byte[] buf = new byte[1024];
DatagramPacket dp = new DatagramPacket
                        (buf, 1024);
ds.receive(dp);
```

```
String str = new String (dp.getData(),
                         0, dp.getLength());
System.out.println (str);
ds.close();
}
}.
```

Q.(3) Explain InetAddress methods with appropriate example.

- This class represents an Internet Protocol (IP) Address.
- The java.net.InetAddress class provides methods to get an IP of host name.
- it is the ~~sub~~ super class of Inet6Address & Inet4Address classes.
- There are no constructors for this class but static methods which returns instances of InetAddress class for general use.

⊛ methods:-

- public static InetAddress getByName (String host) throws ~~Unkow~~ UnknownHostException.

- Determines the IP address of a given host's name.

```
InetAddress ip = InetAddress.getByName
                 ("www.google.com");
System.out.println("ip:" +ip);
```

- public static InetAddress getLocalHost()
  throws UnknownHostException.

- Returns the address of the local host.

```
InetAddress ip = InetAddress.getLocalHost();
System.out.println("LocalHost:" + ip);
```

- public String getHostName()

- it returns the host name of the IP Address

```
InetAddress ip = InetAddress.getByName
                 ("10.254.3.34");
System.out.println("Host Name:" +
                   ip.getHostName());
```

- public String getHostAddress()

- it returns the IP Address in string format

```
InetAddress ip = InetAddress.getByName
                 ("www.google.com");
System.out.println("Host Address:" +
                   ip.getHostAddress());
```

Page No.
Date

Q.(4) Explain URL and URLConnection class with example.

(1) URL class :-

- The Java URL class represents an URL.
- This class is pointer to "resource" on the world wide web.

(2) Example :-

URL url = new URL("http://www.google.com");

(3) method :-

- public URLConnection openConnection() throws IOException

- This method of URL class returns the object of URLConnection class
URLConnection urlCon = url.openCon();

(4) URLConnection class :-

- URLConnection is the superclass of all classes that represent a communication link between the application and a URL.
- Instances of this class can be used both to read from and to write to the resources referenced by the URL.

Page No.
Date

⑦ methods :-

- public InputStream getInputStream()
  throws IOException.

- Returns an input stream that reads from this open connection.

- public OutputStream getOutputStream()

- returns an output stream that writes to this connection.

⑧ Example :-

```
import java.io.*;
import java.net.*;

public class URLConnectionDemo {

  public static void main(String args[]) {

  try {
      URL url = new URL("http://www.google.com");
URLConnection urlcon = url.openConnection();

InputStream stream = urlcon.getInputStream();
int i;
while ((i = stream.read()) != -1) {
      System.out.print((char)i);
  }
  catch (Exception e) { System.out.println(e); }
}
```

Page No.
Date

Q. (5) How to display IPAddress & host name for local machine.

```
import java.net.*InetAddress;

public class Main {

    public static void main(String args[])
        throws Exception {

    InetAddress addr = InetAddress.
                    getLocalHost();
    System.out.println("Local Host Address:"
                + addr.getHostAddress());
    String hostname = addr.getHostName()
    System.out.println("Local host name:"
                + hostname);
```

Page No.
Date

⊛ chapter : ② : JDBC Programming

① what is JDBC ? Explain the types of JDBC Drivers ?

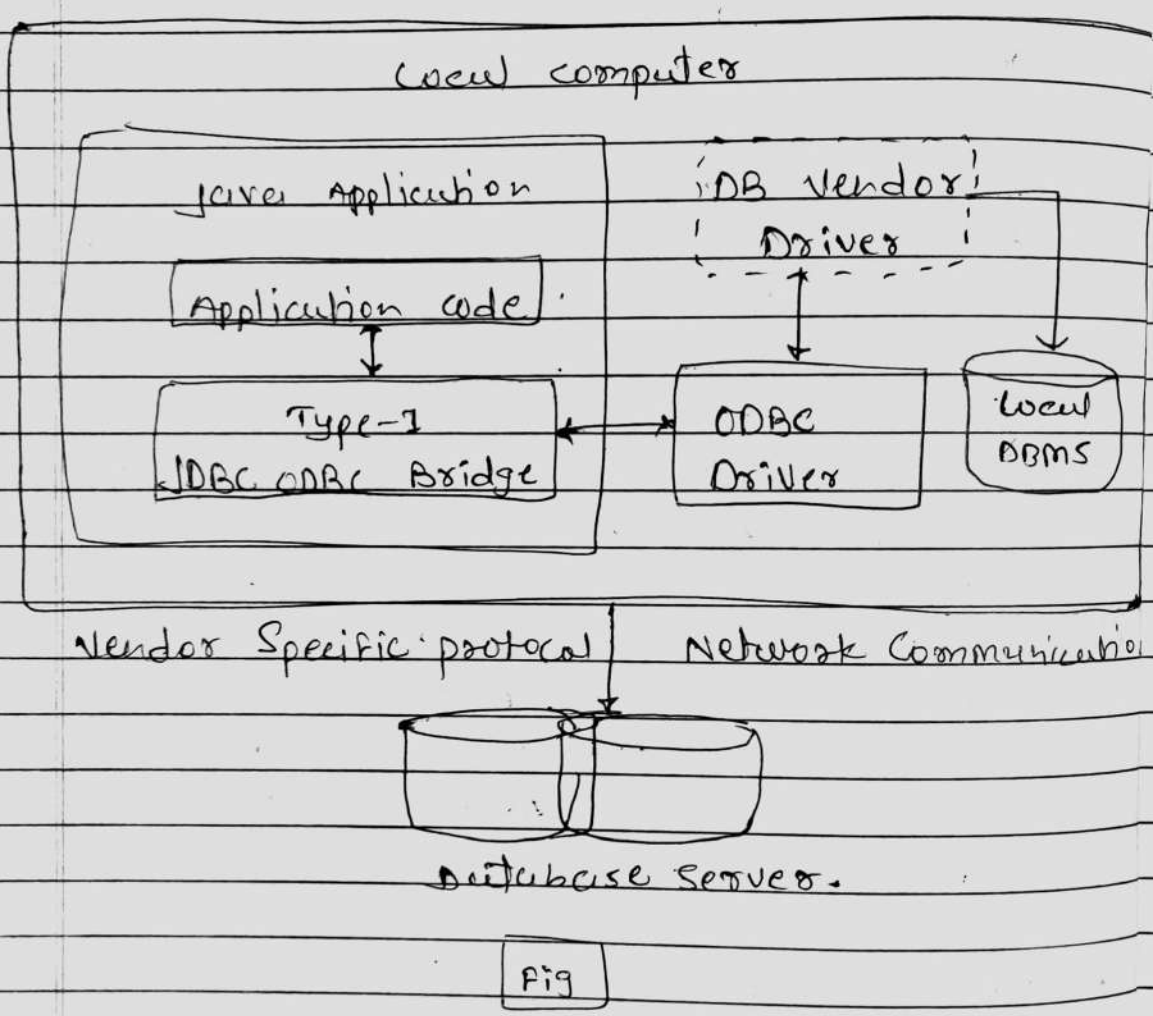
- JDBC is an API, which is used in java programming for interacting with database.
- JDBC (Java database connection) is the standard method of accessing databases from java application.
- JDBC is a specification from sun micro system that provides a standard API for java application to communicate with different database.
- JDBC is a platform independent interface between rational database and java application.

⊛ JDBC drivers : (1) Type-1 (JDBC-ODBC D..)
(2) Type-2 (Native-code Dri..)
(3) Type-3 (Java protocol)
(4) Type-4 (Database protocol)

(1) Type-1 (JDBC-ODBC Drivers) :-

- Depends on support for ODBC
- Type-1 is not portable driver
- Translate JDBC calls into ODBC calls and use windows ODBC built in drivers

- ODBC must be set up on every client
- For server side servlets ODBC must be set up on web browser.
- Driver sun.jdbc.odbc.JdbcOdbc provided by JavaSoft with JDK.
- No support from JDK 1.8 (Java 8) onwards Eg. MS Access.



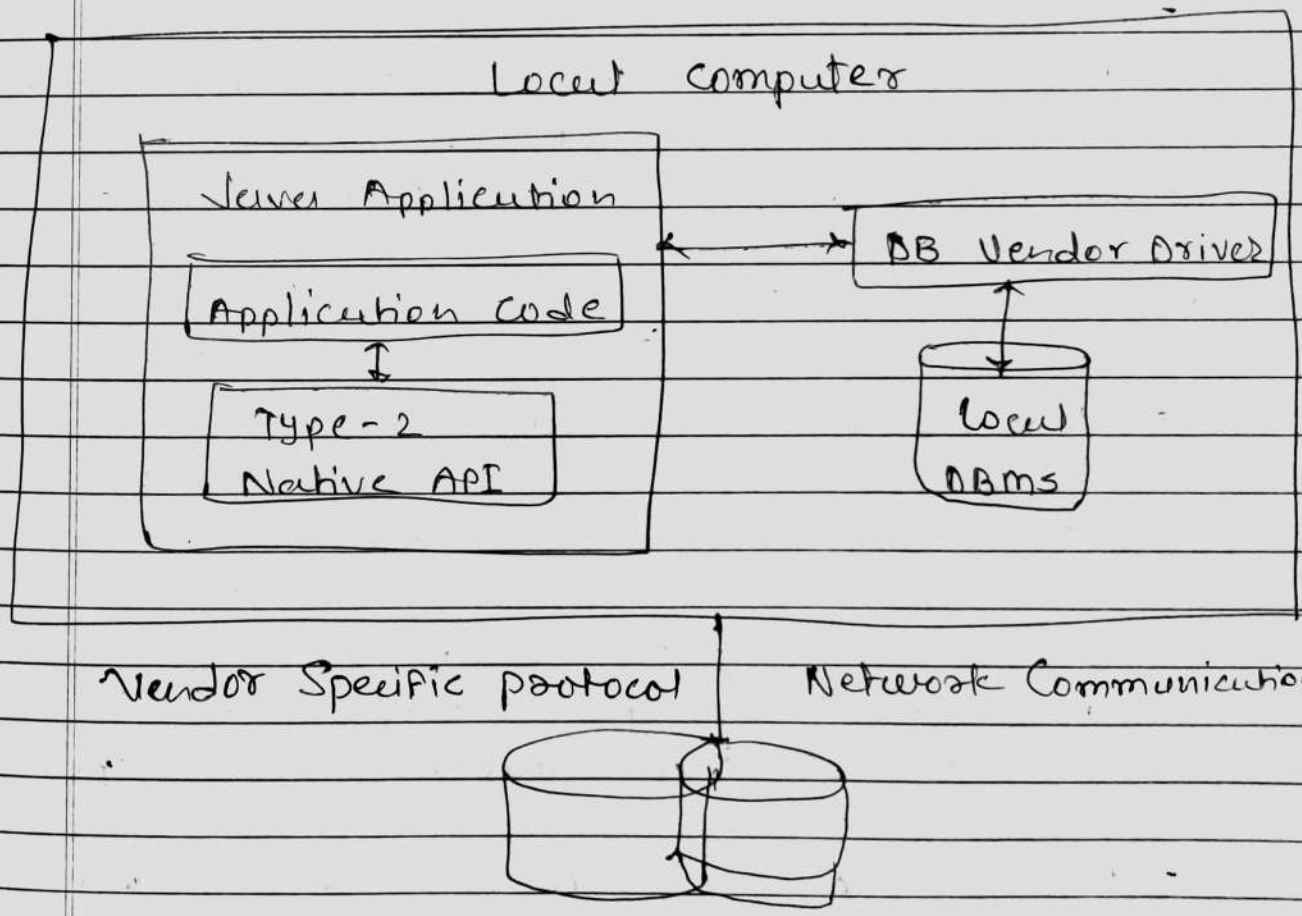
Fig

- Advantages :- Allow to communicate with all databases supported by ODBC driver.
- it is vendor independent driver.

Page No.
Date

② Type-2 (Native Code Driver)

- JDBC API calls are converted into native API calls, which are unique to the database.
- these drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge.
- Native code Drivers are usually written in C, C++.
- the vendor-specific driver must be installed on each client machine.
- Type-2 Driver is suitable to use with server side applications.



Fig

Page No.
Date

(*) Advantages :- As there is no implementation of JDBC-ODBC Bridge, it may be considerably faster than a Type 1 driver.

(*) Disadvantages :-

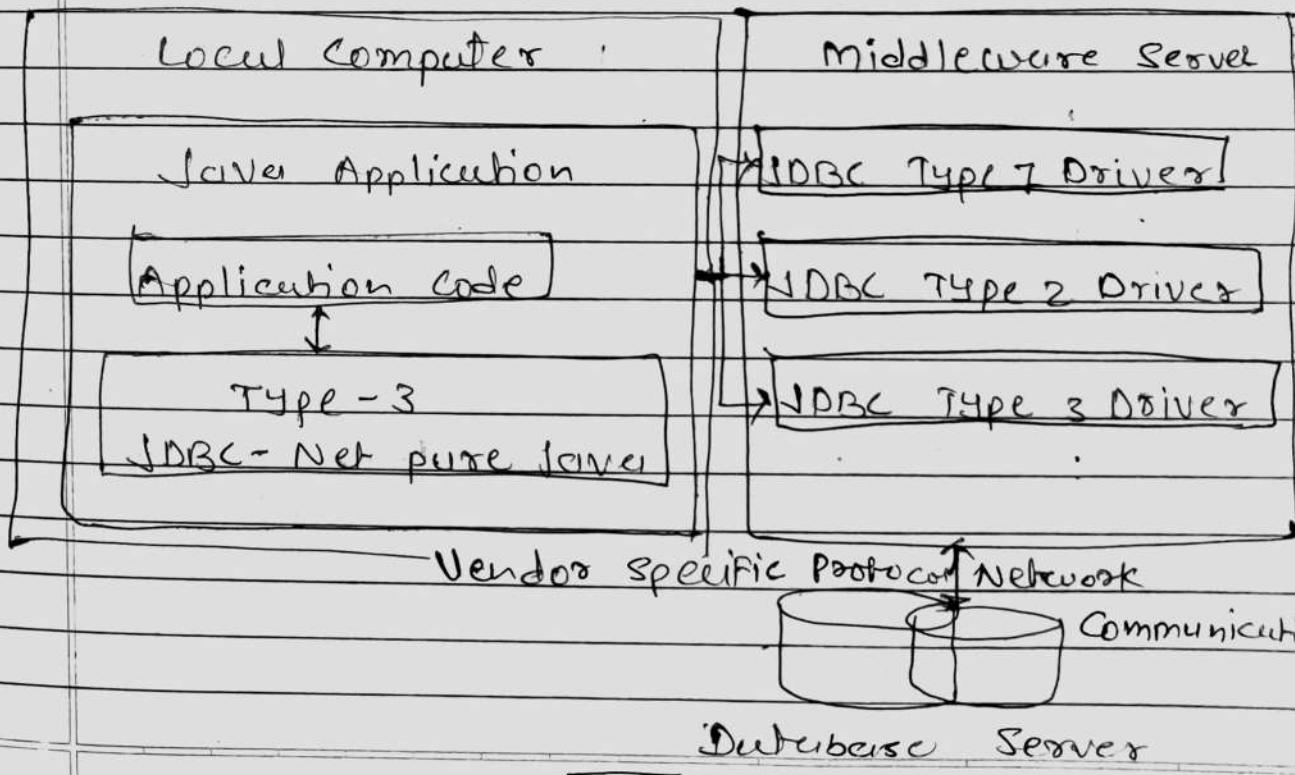
- the vendor client library needs to be installed on the client machine hence type 2 drivers cannot be used for the internet.
- this driver is platform dependent.
- this driver supports all java applications except applets.
- it may increase cost of application, if it needs to run on different platform.
- mostly obsolete now.
- usually not thread safe.

(*) Type-3 (Java Protocol) :-

- this driver translate the jdbc calls into a database server independent and middleware server specific calls.
- with the help of the middleware server, the translated jdbc calls further translated into database server specific.
- this type of driver also known as net-protocol fully java technology-enabled driver.

Page No.
Date

- Type-3 driver is recommended to be used with applets. its auto-downloadable.
- can interface to multiple databases - Not Vendor specific.
- Follows a three-tier communication approach.
- The JDBC clients use standard network sockets to communicate with a 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.



Fig

Page No.
Date

Advantages :-

- Since the communication between client and middleware server is database independent, there is no need for the database vendor library on the client.
- A single driver can handle any database provided the middleware supports it.
- We can switch from one database to other without changing the client-side driver class, by just changing configurations of middleware server.

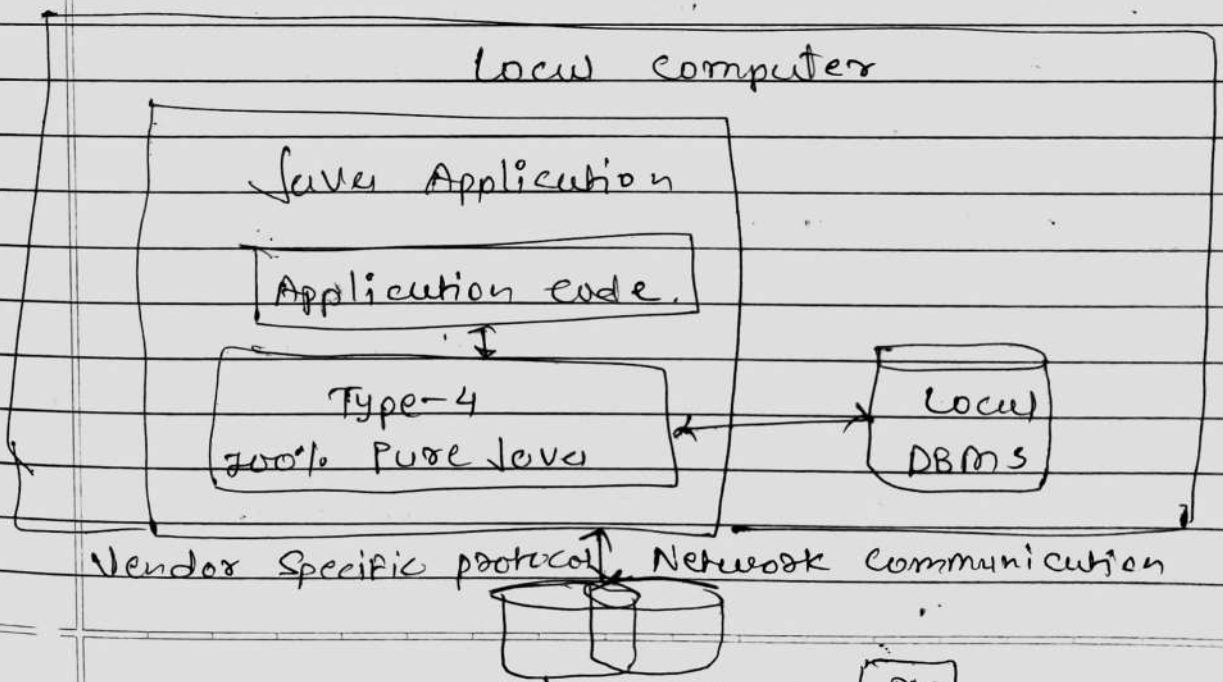
Disadvantages :-

- Compared to Type 2 drivers, Type 3 Drive are slow due to increased number of network calls.
- Requires database-specific coding to be done in the middle tier.
- the middle ~~layer~~ tayer were layer added may result in additional latency, but it typically overcome by using better middleware services.

Page No.
Date

⑧ Type-4 (Database Protocol)

- it is known as the Direct to Database Pure Java Driver.
- Need to download a new driver for each database engine.
- Type-4 driver, a pure Java-based driver communication communicates directly with the vendor's database through socket connection..
- this kind of driver is extremely flexible, you don't need to install special software on the client or server.
- This type of driver is lightweight and generally known as thin driver.
- you can use this driver when you want an auto downloadable option the client side application.



Page No.
Date

(*) Advantages :-

- Completely implemented in Java to achieve platform independence.
- No native libraries are required to be installed in client machine.
- These drivers don't translate the requests into an intermediary format.
- Secure to use since, it uses database server specific protocol.
- The client application connects directly to the database server.
- No translation or middleware layers are used, improving performance.
- The JVM manages all the aspects of the application-to-database connection.

(*) Disadvantage :-

- This Driver uses database specific protocol and it is DBMS vendor dependent.

Q.(2) Explain Thick and Thin driver. Comment on selection of driver. write code snippet for each type of JDBC connection.

* Thick client would the client installation.

Eg. Type 1 and Type 2.

Page No.
Date

④ Thin driver :

- The thin client driver, which mean you can connect to a database without the client installed on your machine.

eg. Type 4.

⑤ Comment on selection of driver :-

- if you are accessing one type of database such as MySQL, Oracle, Sybase or IBM etc. the preferred driver type is 4.

- if your Java application is accessing multiple types of databases at the same time, type 3 is the preffered driver.

- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

- The type 1 driver is not considered a deployment-level driver, and is typically used for ~~dep~~ development and testing purposes only.

⑥ Code snippet for each type of JDBC connection :-

① MySQL / ② Oracle / ③ DB2

① MySQL :-

Class.forName("com.mysql.jdbc.Driver");

Connection conn = DriverManager.getConnection
("jdbc:mysql://localhost:PortNo/databas
Name", "uid", "pwd");

② Oracle :

Class.forName("oracle.jdbc.driver.OracleDrive

Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@hostname:port Number
:databaseName", "root", "pwd");

~~Akrut~~

③ DB2 :

Class.forName("com.ibm.db2.jdbc.net.DB2Drive

Connection conn = DriverManager.getConnection
("jdbc:db2:hostname:port number/
databaseName");

---

③ Explain Statement Interface with appropia example.

- used for general-purpose access to your database.

Page No.
Date

- Useful for static SQL statements, eg. Effect SELECT specific row from table etc.

- The Statement interface denies a standard abstraction to execute the SQL statements requested by a user and return the results by using ResultSet object.

- The statement interface is ~~increased~~ created after the connection to the specified database is made.

- The object is created using the CreateStatement() method of the connection interface, as shown in following code snippet :

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

② program :-

```
import java.sql.*;
public class ConnDemo {
    public static void main(String args[]) {
    try {
        // Load and register the driver
        class.forName("com.mysql.jdbc.Driver");

        // Establish the connection to the database server
        Connection conn = DriverManager.getConnection
"jdbc:mysql://localhost:3306/database_name","root","pwd");
```

```
// Creates a statement.
Statement stmt = conn.createStatement();

// Execute the statement
Statement stmt =
  ResultSet rs = stmt.executeQuery
  ("SELECT * from Table");

// Retrive the results
while (rs.next()) {

    System.out.print(rs.getInt(1) + "\t");
    System.out.print(rs.getString("Name") +
                     "\t");
    System.out.println(rs.getString(3));
} // while

// close the Statement and connection

  stmt.close();
  conn.close();
} catch(Exception e)
  {
System.out.println(e.toString());
} // PSVM
} // class.
```

Page No.
Date

Q.④ Explain Prepared Statement with ex.

- the PreparedStatement interface is subclass of the Statement interface, can be used to represent a precompiled query, which can be executed multiple times.

- Prepared Statement is used when you plan to execute same SQL statements many times.

- Prepared Statement interface accepts input parameters at runtime.

- The object is created using the Prepared Statement() method of connection interface, as shown in following snippet:

```
String query = "insert into emp values (?, ?)";
PreparedStatement ps = con.preparedStatement(query);
ps.setInt(2,5);
ps.setString(2, "New Employee");
int n = ps.executeUpdate();
```

⑤ Advantages:-

- the performance of the application will be faster, if you use PreparedStatement interface because query is compiled only once.

- This is because Creating a PreparedStatement object by explicitly giving the SQL statement causes the statement to be precompiled within the database immediately.

- Thus, when the PreparedStatement is later executed, the DBMS does not have to recompile the SQL statement.

- Late binding and complication is done by DBMS.

- Provides the programmatic approach to set the values.

⊛ Disadvantages :-

- the main disadvantage of Prepared statement is that it can represent only one SQL statement at a time.

⊛ Program :- insert student records to database using Prepared statement.

```
import java.sql.*;
public class PreparedStatement {
    public static void main(String args[]){
    try {
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection
("jdbc:mysql://localhost:3306/DIET", "root", "pwd")
```

```
String query = "insert into dietstudent
    Values (?,?,?,?)";

PreparedStatement ps = conn.preparedStatement
                        (query);

ps.setString (1, "14092"); // enr_no
ps.setString (2, "abc_comp"); // Name
ps.setString (3, "computer"); // Branch
ps.setString (4, "x"); // Division

  int i = ps.executeUpdate();
System.out.println ("no. of rows updated = "
                    + i);
ps.close();
conn.close();
} catch (Exception e) {
System.out.println (e.toString());
}
} // psvm
} // class.
```

Q. (5) Explain Callable Statement with example.

- CallableStatement interface is used to call the stored procedures.
- Therefore, the stored procedure can be called by using an object of the callableStatement interface.

- The object is created using the prepare Call Method of Connection interface.

```
CallableStatement cs = conn.prepareCall
                    ("{call proc_Name(?,?)}");
cs.setInt(1, 2222);
cs.registerOutParameter(2, Types.VARCHAR);
cs.execute();
```

- Three Types of parameters exist:
  IN / OUT and INOUT.

- PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

ⓐ parameters:

-① - IN:

- A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.

-② OUT:

- A parameter whose value is supplied by the SQL statement it returns. You retreive values from the OUT parameter with the getXXX() methods.

Page No.
Date

③ - INOUT :

- A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

① program :-

- write a callable statement program to retrieve branch of the student using {getBranch() procedure} from given enrollment number. Also write code for stored procedure. [Stored Procedure : ↓ getBranch().]

```
1. DELIMITER @@
2. DROP PROCEDURE getBranch @@
3. CREATE PROCEDURE databaseName.getBranch
4. (IN enr_no INT, OUT my_branch
                    VARCHAR(10))
5. BEGIN
6. SELECT branch INTO my_branch
7. FROM dietstudent
8. WHERE enr_no = enrno;
9. END @@
10. DELIMITER;
```

② Callable Statement program :-

```
import java.sql.*;
public class CallableDemo{
public static void main (String args[]){
try{
  Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection
("jdbc:mysql://localhost:3306/Diet","root","pwd");

CallableStatement cs = conn.prepareCall
                    ("{call getBranch(?,?)}");
cs.setInt(1,2222);
cs.registerOutParameter(2,Types.VARCHAR);
cs.execute();
System.out.println("branch=" + cs.getString(2));
cs.close();
conn.close();
}catch(Exception e){
  System.out.println("e.toString());
}
} // psvm
} // class.
```

Q. (7) Explain JDBC Architecture.

(A) JDBC API :

- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

Page No.
Date

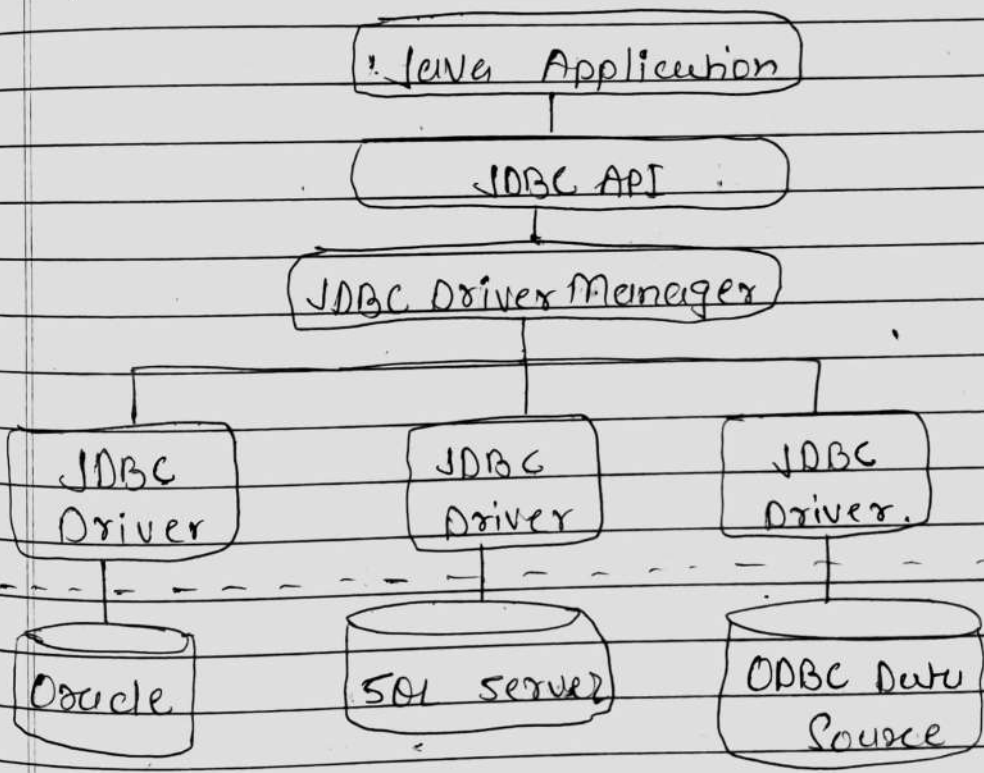
- JDBC API Provides classes and interfaces to connect or communicate Java application with database.

- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers -

① JDBC API : This provides the application-to-JDBC Manager connection.

② JDBC Driver API : This supports the JDBC Manager-to-Driver Connection.

③ JDBC Architecture Figure:-



Fig

Page No.
Date

(x) JDBC Driver Manager (Class):

- This class manages a list of databases
- it ensures that the correct driver is used to access each data source.
- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.
- matches connection requests from the java application with the proper database driver using communication sub protocol.
- The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database connection.

(x) Driver (Interface):-

- This interface handles the communications with the database server.
- You will interact directly with Driver objects very rarely.
- Instead, you use DriverManager objects, which manages objects of this type.
- It also abstracts the details associated with working with Driver objects.

(x) Connection (Interface):-

- This interface with all methods for contacting a database.
- The connection object represents communication context, i.e, all communication with database is through connection object only.

Page No.
Date

① Statement (Interface):-

- you use objects are created from this interface to submit the SQL statement to the database.
- some derived interfaced accept parameters in addition to executing store procedures.

② ResultSet (Interface):-

- these objects hold data retrieved from a database after you execute on SQL Query using Statement objects.
- It acts as an iterator to allow you to move through its data.

③ SQLException (class):-

- This class handles any errors that occur in a database application.

④ Differentiate executeQuery()/Update()/ & execute().

① executeQuery():

- ResultSet executeQuery(String SQL) throws SQLException

- This is used generally for reading the content of the database. The output will be in the form of ResultSet.

- Generally SELECT statement is used.

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

② executeUpdate() :-

- int executeUpdate (String sql) throws SQLException

- This is generally used for altering the databases.

- Generally DROP, INSERT, UPDATE, DELETE statements will be used in this.

- The output will be in the form of int.
- This int value denotes the number of rows affected by the query.

Eg. int i = stmt.executeUpdate (query);

③ execute() :-

- Boolean execute (String sql) throws SQLException.

Page No.
Date

- if you dont know which method to be used for executing SQL statements, this method can be used.

- this will return boolean. TRUE indicates the result is ResultSet and FALSE indicates it has the int value which denotes number of rows affected by the query.

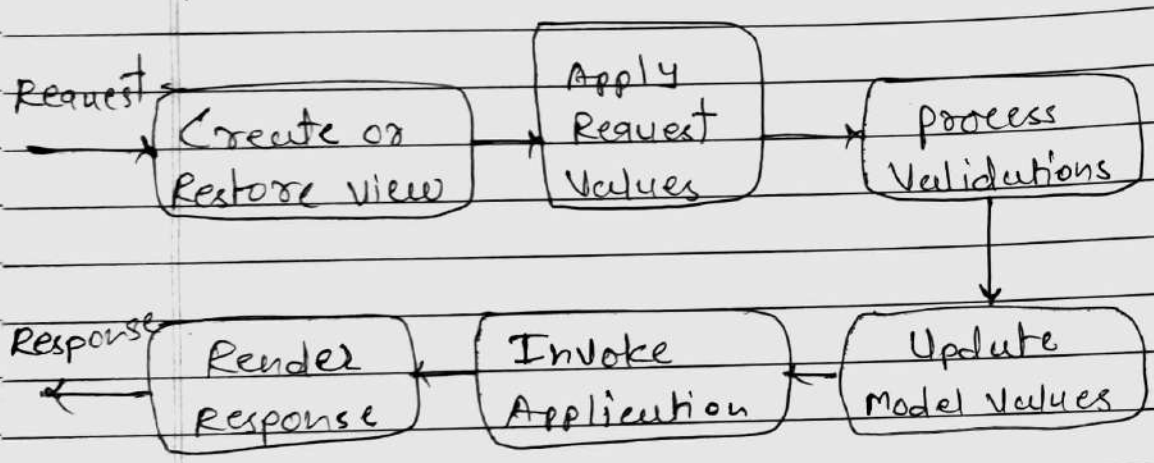
Eg. Boolean b = Stmt.execute (query);

⊛ Chapter : ⑤ : Java Server Faces ⊛

Q.① JSF Request processing lifecycle and function of each phase.

- JSF application lifecycle consist of six phases which are as follows:

phase : 1 : Restore View (RV)
phase : 2 : Apply Request Values (ARV)
phase : 3 : process Validations (PV)
phase : 4 : Update Model Values (UMV)
phase : 5 : Invoke Application (IA)
phase : 6 : Render Response (RR)



⊛ phase : 1 : Restore View :-

- JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request.

- During this phase, the JSF build the

view, wires event handlers & validators to UI components and saves the view in the FaceContext instances.

- the FaceContext instance will now contains all the information required to process a request.

⊛ phase : 2 : Apply request values :-

- In this phase, the values that are entered by the user will be updated on each and every individual component defined in the view graph.

- Component stores this value.

- If any of the Conversions or the Validations fail, then the current processing is terminated and the control directly goes the Render Response for rendering the conversion or the validation errors to the client.

⊛ phase : 3 : Process Validation :-

- This phase will process any validations that are configured for UI components.

- These validations will only happen for the UI components only if the property 'rendered' property is set to 'true'.

④ phase : 4 : Update model values :-

- After the JSF checks that the data is valid, it walks over the component tree and set the corresponding server-side object properties to the component's local values.

- The JSF will update the bean properties corresponding to input component's value attribute.

⑤ phase : 5 : Invoke application :-

- During this phase, the JSF handles any application-level events, such as submitting a form / linking to another page.

- Instances MB (Managed Bean), adds value of component to properties of MB, method of MB is executed.

⑥ phase : 6 : Render response :-

- And finally, we have reached the Render Response whose job is to render the response back the client Application.

Page No.
Date

Q. ② JSF Standard Components :

(1) h:inputText -

- HTML input of type = "text"
<h:inputText id = "username" value = "" />

(2) h:inputSecret -

- HTML input of type = "password"
<h:inputSecret id = "password" value = "" />

(3) h:inputHidden -

- HTML input of type = "hidden"
<h:inputHidden value = "Hello world"
id = "hiddenField" />

(4) h:selectMany Checkbox -

- A group of HTML check boxes

<h:selectManyCheckbox value = "" >
<f:selectItem itemValue = "1" .
itemLabel = "Diet J2SE" />
<f:selectItem itemValue = "2"
itemLabel = "Diet J2EE" />
</h:selectManyCheckbox >

(5) h:selectOneRadio -

```
<h:selectOneRadio value="Semester"
  <f:selectItem itemValue="7"
    itemLabel="sem 4" />
  <f:selectItem itemValue="2"
    itemLabel="sem 6" />
</h:selectOneRadio>
```

(6) h:outputText -

- HTML text
- `<h:outputText value="username:" />`

(7) h:commandButton -

- HTML input of type="submit" button.

(8) h:Link -

- HTML anchor.

`<h:link value="page 7" outcome="page7" />`

Q.(3) JSF Facelets :-

- JSF provides special tags to create common layout for a web application called Facelets tags.
- These tags provide flexibility to manage common parts of multiple pages at one place.

- Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces views using HTML style templates and to build component trees.
- For these tags, you need to use the following namespaces of URI in html page.

```
<html
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:ui = "http://java.sun.com/jsf
                /facelets" >
```

⊛ Facelets tags :

(1) ui : insert -

- Insert content into a template. That content is define with the ui:define tag.

(2) ui:define -

- The define tag defines content that is inserted into a page by a template.

(3) ui: Composition -

- The <ui:composition> tag provides a template encapsulating the content to be included in the other Facelets.

(4) ui: include-

- This tag includes the component in the src attributes as a part of the current JSF pages.

⊗ Facelets Features :-

- Use of XHTML for creating web pages.
- Faster than using jsp.
- No tld files and no tag classes to defined a UIComponent.
- .xhtml instead of .jsp.

⊗ Advantages of facelets :-

- ~~Support for~~ Faster compilation time
- Compile-time EL validation.
- High-Performance rendering.

Ex.

```
<html>
  <h:body>
    <div id="top" class="top">
      <ui:insert name="top">
      Top Section
      </ui:insert>
    </div>
    <ui:define name="top">
    welcome to Template client page
    </ui:define>
  </h:body>
</html>
```

Page No.
Date

Q. ④ JSF Validation tags :-

- JSF provides inbuilt validators its UI components. These tags can validate the length of the field, the type of input which can be a custome object.

- For these tags you need to use the following namespaces of URI in html code.

```
<html
    xmlns = "http://www.w3.org/1999/xhtml">
    xmlns:f = "http://java.sun.com/jsf
                /core" >
```

④ Validation tags :-

(1) f:validateLength :

- Validates length of a string

```
<f:validateLength minimum = "10"
                  maximum = "15" />
```

(2) f:validateLongRange -

- Validate range of numeric value

```
<f:validateLongRange minimum = "10"
                     maximum = "15" />
```

(5) f:validateDoubleRange -

- Validates range of float value. />

<f:validateDoubleRange minimum = "100.00" maximum = "1000.

(4) f:validateRegex -

- Validates JSF component with a given regular expression.
- <f:validateRegex pattern = "((?=.*[a-z]).{6,})" />

(5) Custom Validator -

- Creating a custom validor.

- Create a validor class by implementing javax.faces.validator.Validator interface.
- Implement validate() method of above interface.
- Use Annotation @FacesValidator to assign a unique id to the custom validator.

Q.(8) JSF Expression Language.

- JSF provides a rich expression language we can write normal operations using #{operation-expression} notation.

- Some of the advantages of JSF Expression languages are following!

(1) provides easy access to elements of a collection which can be a list, map or an array.

(2) provides easy access to predefined objects such as request.
(3) Arithmetic, logical, relational operations can be done using expression language.
(4) Automatic type conversion.
(5) shows missing values as empty strings instead of NullPointerException.

Notation:

#{operation - expression}

Ex: #{1=70} <!-- output: 70 -->
#{10>9} <!-- output: true -->
#{'userCurrent'.add} <!-- will calls add() of userCur bean -->

Q.(6) PrimeFaces :-

- PrimeFaces is a lightweight library with one jar, zero-configuration and no required dependencies.

- To use JSF prime faces library we need to use following code:

<html
xmlns = "http://www.w3.org/1999/xhtml"
xmlns:p = "http://primefaces.org/ui">

Page No.
Date

⑧ Characteristics of Prime Faces :-

(1) Simplicity and Performance :

- Primefaces is a lightweight library, all decesions made are based on keeping prime faces as lightweight as possible.

- Usually adding a third-party solution could bring a overhead however this is not the ease with Prime Faces.

- it is just one single jar with no dependencies and nothing to configure.

(2) Easy to use :

- components in Primefaces are developed with a design principle which states that "A good UI component should hide complexity but keep the flexibility

(3) Strong Community feedback :

- Primefaces community continuously helps the development of Primefaces by providing feedback, new ideas, bug reports and patches.

Page No.
Date

(2) JSF prime faces provides following collection of tags:

(1) <p: inputText>
(2) <p: inputSecret>
(3) <p: commonButton>
(4) <p: commond Link>
(5) <p: ajax>
(6) <p: beorchust>
(7) <p: calendar>
(8) <p: colorPicker>
(9) <p: dialog>
(10) <p: FileUploud>
(11) <p: FileDownloud>

11.

Chapter : 4 : Java Server Pages

Q.1 List and Explain various stages of JSP life cycle. Briefly give the function of each phases.

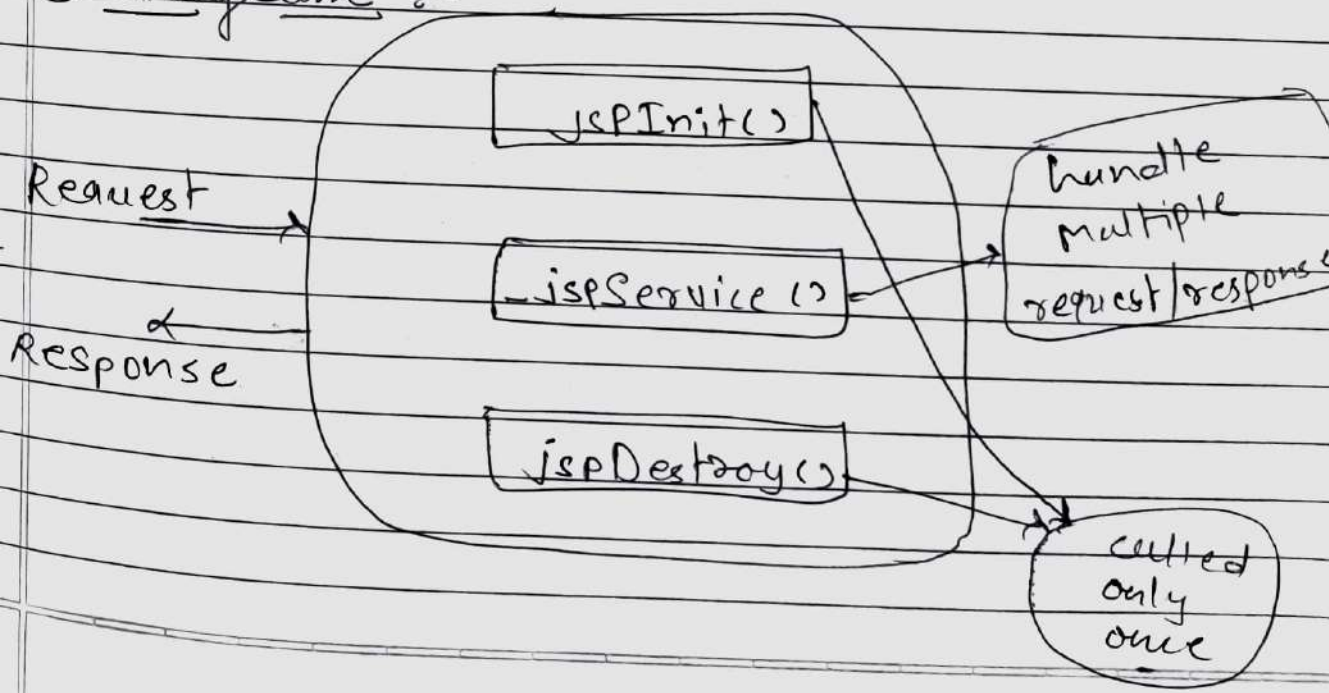
1. A JSP life cycle can be defined as the entire process from its creation till the destruction.

2. It is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

3. A JSP page is converted into servlet in order to service requests.

4. The translation of a JSP page to a servlet is called life cycle of JSP.

diagram :-



In detail - Darshan Notes
↳ pg.no/77/78.

Page No.
Date

④ JSP Lifecycle stages :-

(1) Translation of JSP to servlet code
(2) Compilation of Servlet to bytecode.
(3) Loading servlet class
(4) Creating servlet instance
(5) Initialization by calling jspInit() method

```
public void jspInit()
{
// Initializing the code
}
```

(6) Request Processing by calling _jspService() method.
(7) Destroying by calling jspDestroy() method.

Q.②	JSP	Servlet
(1)	JSP is a webpage scripting language that generates dynamic content	Servlets are Java programs that are already compiled which also creates dynamic web content
(2)	A JSP technically gets converted to a Servlet we embed the Java code into HTML.	A servlet is a Java class. we can put HTML into print statements.
	eg. <html> <% Java code %> </html>	eg. out.println ("<html code>");

(3) JSPs are extensions of servlets which minimizes the effort of developers to write User Interfaces using Java programming.	A servlet is a server-side program and written purely on Java.
(4) JSP runs slower than Servlet. As, it has the transition-phase for converting from JSP to a servlet. Once it is converted to a Servlet then it will start the compilation.	Servlets runs faster than JSP.
(5) In MVC Architecture JSP acts as view	In MVC Architecture Servlet act as controller.
(6) We can build custome tags using JSP API.	We cannot any custome tags in servlet.

8. (8) Advantages of JSP over Servlets.

(1) JSP needs no compilation. There is automatic deployment of a JSP, recompilation is done automatically when changes are made to JSP.

(2) In a JSP page visual content & logic are seperated, which is not possible in a servlet.

i.e JSP seperates business logic from the presentation logic.

(3) Servlets use println statements for printing an HTML document which is usually very difficult to use. JSP has no such tedious task to maintain

Q. (4) JSP Scripting Elements :-

- The scripting elements provides the ability to insert java code inside the jsp. There are three types of traditional scripting elements:

(1) Scriplet tag
(2) Expression tag
(3) Declaration tag.

[1] Scriplet tag :-

- A scriplet tag is used to execute java source code in jsp.

- A scriplet tag can contain.
  - Any number of java language st
  - Variables - Method declarations
  - Expressions

Page No
Date

Syntax :-

<% // java source code %>

Ex : <% out.print("welcome to jsp"); %>
<% int a=10; %>

- Everything written inside the scriplet tag is compiled as java code.

- jsp code is translated to servlet code, in which _jspService() method is executed which has HttpServletRequest and HttpServletResponse as argument.

- jsp page can have any number of scriplets, and each scriplets are appended in _jspServices.

[2] Expression tag :-

- the code placed within jsp expression tag is written to the output stream of the response.

- So you need not to print the values of variable or method.

- it is mainly used to print the values of variable or method.

- Do not end your statement with semicolon in case of expression tag.

Page No.
Date

Syntax :-

<%= statement %>

Ex: <%= (2*5) %>

[3] Declaration tag :-

- The JSP declaration tag is used to declare variables and methods.
- The declaration of jsp declaration tag is placed outside the _jspService() method.

Syntax :-

<%! variable or method declaration %>

Ex:

<%! int a = 10; %>
<%! int a, b, c; %>
<%! Circle a = new Circle (2.0); %>

[4] Comments :-

- The comments can be used for documentation.
- This jsp comment tag tells the JSP container to ignore the comment part from compilation.

Page No.
Date

Syntax :-

<%-- comments --%>

jsp comment - <%-- jsp comment --%>
Java comment - /* multi line */
// single line
Html comment - <!-- html comment -->

Q.(5) JSP Page Directives :-

- JSP directives provide directions & instructions to the container, telling it how to translate a JSP page into the corresponding servlet.

- A JSP directives affects the overall structure of the servlet class.

- JSP engine handles directives at translation time.

- There are two types of directives:

(1) page directive
(2) include directive.

Syntax: <%@ directive
attribute = "value" %>

(1) Page directive :-

- The page directive defines that apply to an entire JSP peges.
- You may code page directives anywhere in your JSP page.
- By convention, page directives are coded at the top of the JSP page.

Syntax :

<%@ page attribute = "value" %>

Ex :

<%@ page import = "java.util.Date, java.util.List, java.io.* "%>

<%@ page contentType = "text/html"; charset = US-ASCII "%>

(*) Attributes of Page directives :-

(1) import
(2) contentType
(3) extends
(4) info
(5) buffer
(6) language
(7) isELIgnored
(8) autoFlush
(9) session
(10) errorPage
(11) isErrorPage.

(1) import :

- Used to import class, interface or all the members of a package.

<%@ page import = "java.util.Date" %>
Today is : <%= new Date() %>

(2) contentType :

- The contentType attributes defines the MIME type of the HTTP response. The default value is "text/html; charset = ISO-8859-1".

<%@ page contentType = application/msword %>

(3) extends :

- The extends attribute defines the parent class that will be inherited by the generated servlet.

<%@ page extends = "javax.servlet.HttpServlet" %>

(4) info :

- This attribute sets simply sets the information of the JSP page which is retrieved later by using getServletInfo().

<%@ page Info = "Authored by : Author Name" %>

(5) buffer :

- The buffer attribute sets the buffer size in kb to handle output generated by the JSP page.

- The default size of the buffer is 8kb.

<%@ page buffer = "16kb" %>

(6) language :

- The language attribute specifies the scripting language used in the JSP page. The default value is "java".

- ~~The default size of the~~

<%@ page language = "java" %>

(2.) JSP Include directive :-

- JSP include directive is used to include the contents of another file to the current JSP page during translation time.

- Include directive is used for merging external files to the current JSP page during translation phase.

- the include file can be HTML, JSP, text files, etc.

② Advantages of Include directive :-

- Code Reusability

Syntax :

<%@ include attribute = "value" %>

Ex :

<%@ include file = "1.jsp" %>

Q. ③ JSP Implicit Objects :-

∴ There are 9 Implicit objects.

- these objects are created by web container that are available to all the jsp pages.

		implicit object of
(1)	out	-
(2)	request	javax.servlet.http.HttpServletRequest
(3)	response	" " " .HttpServletResponse
(4)	config	javax.servlet.ServletConfig.
(5)	session	javax.servlet.http.HttpSession.
(6)	pageContext	javax.servlet.jsp.PageContext.
(7)	page	java.lang.Object.
(8)	application	javax.servlet.ServletContext
(9)	exception	java.lang.Throwable.

(1) out :-

- For writing any data to the buffer, JSP provides an implicit objects named out.

- it is an object of JspWriter.

```
<html>
<body>
    <% out.print("DIET"); %>
</body>
</html>
```

(2) request :-

- the request object provides methods to get HTTP headers information including from data, cookies, HTTP methods etc.

```
<% out.println(request.getParameter
                ("login")); %>
```

(3) response :-

- Through this object the JSP programmer can add new cookies or data stamps, HTTP status codes, redirect response to another resources, send error etc.

```
<% response.sendRedirect("www.rohan.com"); %>
```

(4) config :-

- this object can be used to get initialization parameter for a perticular JSP page.

<%! out.print("welcome" + request.getParameter("login")); %>

String c-name = config.getInitParameter("college");
out.print(" College name is = " + c-name + " "); %>

(5) session :-

- this object is used to set, get or remove attribute or to get session information.

(6) pageContext :-

- this object is used to set, get or remove attribute.

(7) page :-

- this object is an actual reference to the instance of the page.
- it
  ex. returns the name of generated servlet files.

<% out = page.getClass().getName() %>

(8) application :-

- This object is used to get initialization parameter from configuration file (web.xml).

- This initialization parameter can be used by all jsp pages.

<%. // refers to context parameter of web.xml

String driver = application.getInitParameter("name");
out.print("name is =" + name); %>

(9) execution :-

- This object is used to print the exception. But it can only be used in error pages.

<%@ page isErrorPage = "true" %>

<html> <body>
Exception occured:
<%= exception %>
</body>
</html>

Page No.
Date

Q. ⑦ JSP Action Elements :-

- JSP actions use constructs in XML syntax to control the behaviour of the servlet engine.

- We can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the java plugin.

Syntax :

< jsp : action-name attribute = "value" / >

- There are four types of JSP Action Elements.

(1) < jsp : param > :-

- This action is useful for passing the parameters to other JSP action tags such as JSP include & JSP forward tag.

- This way new JSP pages can have access to those parameters using request object itself.

Syntax :

< jsp : param name = "name"
value = "value" />

Page No.
Date

Ex.

<jsp:param name="date" value="10-03-2023"
<jsp:param name="time" value="10:15 AM"
<jsp:param name="date" value="ABC" />

(2) <jsp:include> :-

- The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet.

- The jsp:include tag can be used to include static as well as dynamic pages

* Attributes of <jsp:include> Action tag:

(1) page:

- The relative URL of the included page to be

(2) Flush:

- The boolean attribute determines whether the included resource has its buffer flushed before it is included.

- By default value is false.

Syntax :

```
<jsp:include page="relative URL"
        flush="true" />
```

Ex. :

```
<jsp:include page="2.jsp" />
```

(3) <jsp:forward> :-

- Forwards the request and response to another resource.

Syntax :

```
<jsp:forward page="Relative URL" />
```

Ex :

```
<jsp:forward page="2.jsp" />
```

(4) <jsp:plugin> :-

- This tag is used when there is a need of a plugin to run a Bean class or an Applet.

- The <jsp:plugin> action tag is used to embed applet in the jsp file.

- The <jsp:plugin> action tag downloads at client side to execute an applet or beun.

Syntax:

```
<jsp:plugin  type="applet | beun"
             code="nameOfclassFile"
             codebase="URL" />
```

Ex. MyApplet.java

```
import java.applet.*;
import java.awt.*;

public class MyApplet extends Applet
{
public void paint(Graphics g)
  {
g.drawString("welcome in java
              Applet.",40,20);
  }}
```

MyPlugin.jsp

```
<html><body>
  <jsp:plugin
        type="applet"
        code="MyApplet.class"
        codebase="/jspclass/
                  MyApplet" />
</body>
</html>
```

Q. (8) What is EL Scripting? Explain EL implicit object and EL Operator. 6

- EL stands for Expression Language Scripting.
- The Java Expression language is a Special purpose programming language. Also mostly used in java web applications for embedding expressions into webpages
- it is newly added feature in jsp technology version 2.0.
- The purpose of EL is to produce Scriplet less JSP pages.

Syntax :

${ expr }

Ex :

EL	output
${a=10}	10
${10+20}	30
${20*2}	40
${10==20}	false
${'a'<'b'}	true

Page No.
Date

⑧ EL Implicit objects :-

(1) pageScope - it is used to access the value of any variable which is set in the perge scope.

(2) requestScope - it is used to access the value of any variable which is set in the request ~~sop~~ scope.

(3) SessionScope :- it is used to access the value of any variable which is set in Session scope.

(4) applicationScope - it is used to access the value of any variable which is set in the Application scope.

(5) pageContext - it represents the pageContext object.

(6) param - map a request parameter ~~value~~ name to a single value.

(7) paramValues - map a request parameter name to corresponding array of string values.

(8) header - map containing names & single string values.

(9) headerValues - map containing names same.

(10) Cookie - map containing cookie names and string values.

Page No.
Date

Q.9 Exception Handling in JSP.

- JSP provides three different ways to perform exception handling.

(1) using simple try.... catch block.
(2) using isErrorPage and errorPage attribute of page directive.
(3) using <error-page> tag in Deployment Descriptor.

---

[1] Using try...catch block is just like how it is used in core Java.

Ex.

```
<html>
<body>
<%
try {
    int i = 100;
    i = i/0;
    out.print("the answer is"+i);
}
catch (Exception e){
out.println("An Exception occurred :"
+ e.getMessage());
}
%>
</body>
</html>
```

(2) Using isErrorPage & errorPage attribute of page directive.

Ex: 1.jsp

```
<%@ page errorPage="2.jsp" %>
<% int i=10;
   i=i/0; %>
```

2.jsp

```
<%@ page isErrorPage="true" %>
<html><body>
  An Exception had occured
<% out.println(exception.toString()); %>
</body></html>
```

(3) Using <error-page> tag in Deployment Descriptor.

- Declaring error page in Deployment Descriptor for entire web application.
- Specify Exception inside <error-page> tag in the Deployment Descriptor.
- we can even configure different error pages for different exception types, or HTTP error code type (503, 500, etc).

Page No.
Date

Ex.
```
<error-page>
    <error-code> 404 </error-code>
    <location> /error.jsp </location>
</error-page>
<error-page>
    <error-code> 500 </error-code>
    <location> /error.jsp </location>
</error-page>
```

Q.(18) JSTL :-

- The core group of tags are the most Frequently used JSTL tags.

- The JSTL core tags provides Variable Support, URL Management, Flow control, etc.

(A) Classification of JSTL tags :-

(1) Core tags
(2) Formatting tags
(3) SQL tags
(4) XML tags
(5) JSTL Functions

Darshan Notes :

Pg.no 94 | to | 99.

Page No.
Date

Q.(11) Customs tag :-

- A custome tag is a user-defined JSP language element.

- When a JSP page containing a custome tag is translated into a servlet, the tag is converted.

- The WebContainer then invokes those operations when the JSP page's servlet is executed.

- JSP tag extensions let you create new tags that you can insert directly into a Java server page just as you would the built-in tags.

(1) To create a custome tag we need three things :

(1) Tag handle class :

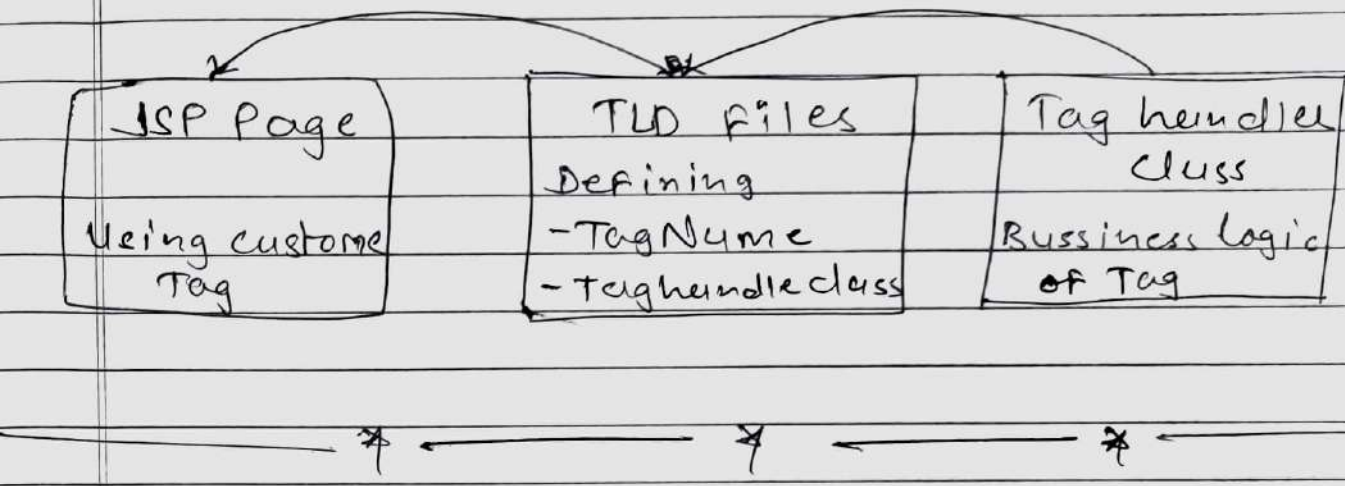
- In this class we specify what our custome tag will do, when it used in a JSP page.

(2) TLD files :

- Tag descriptor file where we will specify our tag name, tag handler class and attributes.

(3) JSP page :

- A JSP page where will be using our custome tag.



⊛ chapter : ③ : Servlet API Overview ⊛

Q.① What is servlet? List & Explain various stages of servlet life cycle. Explain the role of web container.

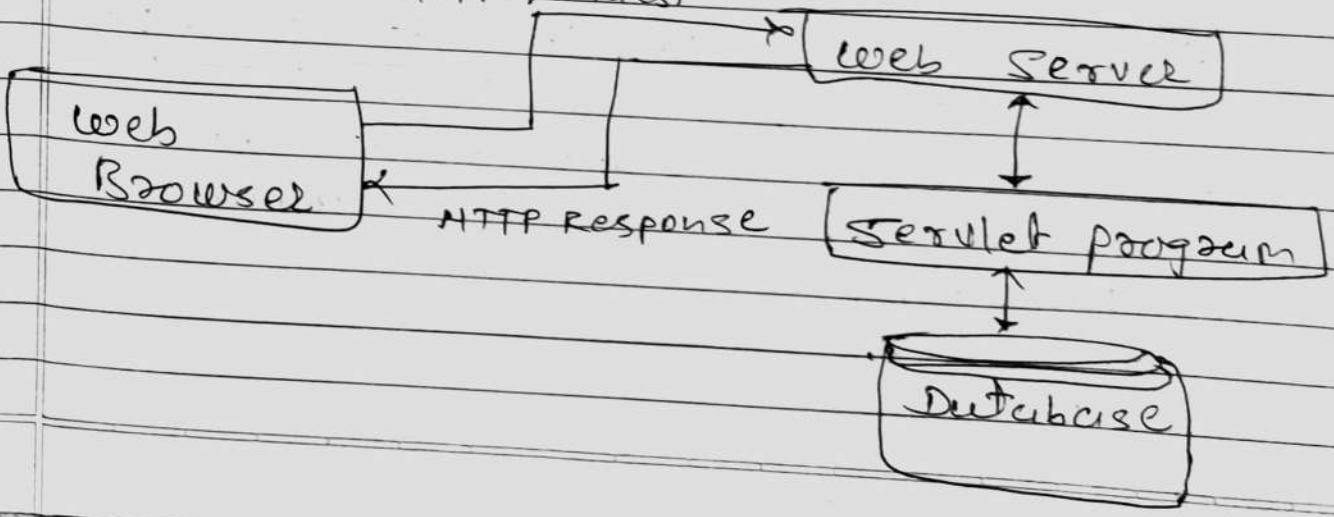
- servlets is are the Java programs that run on the Java-enabled web server or application server.

- they are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

⊛ properties of Servlets :

(1) Servlets work on the server-side.
(2) Servlets are capable of handling complex requests obtained from the webserver.

⊛ Servlet Architecture :-



Page No.
Date

(2) Execution of servlets basically involves six basic steps:

(1) The client send the request to the web ~~browser~~. server.

(2) The web ~~browser~~ server receives the request.

(3) The web server passes the request to the corresponding servlet.

(4) The servlet processes the request and generates the response in the form of output.

(5) The servlet sends the response back to the web server.

(6) The web server send the response back to the client and the client browser display it on the screen.

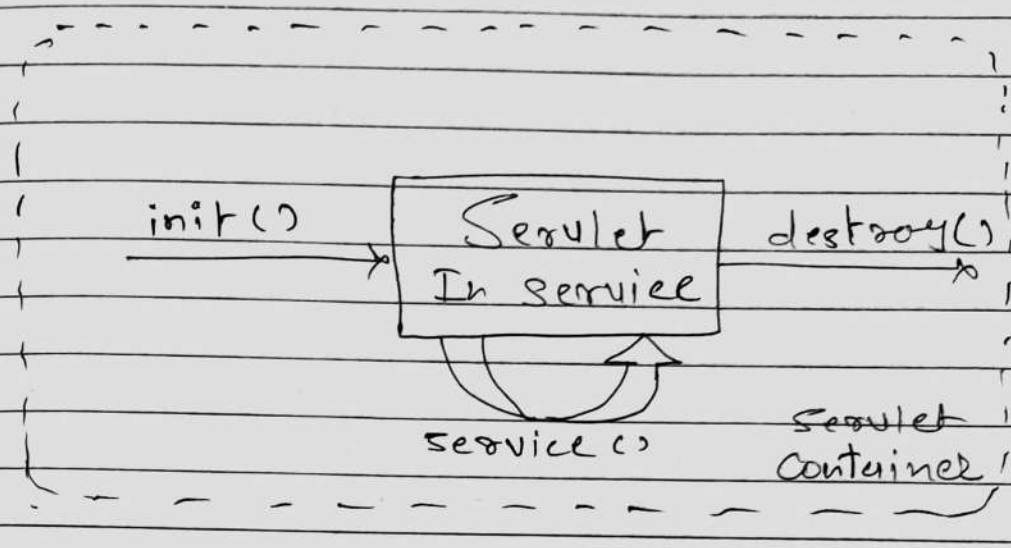
(3) Stages of Servlet Life cycle:-

- in the life cycle of servlet there are three ~~main~~ important methods.

(1) init()
(2) service()
(3) destroy()

Page No.
Date

⊛ diagram :-



- the clients enters the URL in the web browser and make a request. the browser then generates the HTTP Request and sends it to the web server.

- Web server map this request to the corresponding servlet.

(1) init() :-

- the server basically invokes the init() method of servlet. this method is called only when the servlet is loaded in the memory for the first time.

- the class loader 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.

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

- The web container calls the init method only after creating the servlet instance. The init() method is used to initialize the servlet.

```
public void init ( ServletConfig
                   config )
    throws ServletException
{
    // servlet Initialization...
}
```

- A servlet configuration object used by a servlet container to pass information to a servlet during initialization.

(2) Service() :-

- The service() method is the main method to perform the actual task.

- The servlet container (i.e web server) calls the service() method to handle requests coming from the client (browsers) and to write the response back to the client.

- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.

```
public void service (ServletRequest
request, ServletResponse response)
    throws ServletException, IOException
{
    // servlet Task
}
```

(3) destroy() :-

- Finally server unloads the servlet from memory using the destroy() method.

- The destroy() method is called only at the end of the life cycle of a servlet.

- This method gives your servlet a chance to close

(1) database connections,
(2) halt background threads,
(3) write cookie lists or hit counts (to disk)
(4) perform other such cleanup activities.

- After the destroy() method is called, the servlet object is marked for garbage collection.

```
public void destroy()
{
   // finalization code...
}
```

Example :-

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

public class MyServlet extends
        GenericServlet
  {
public static void init()
  throws ServletException
{ // Initialization code...
}
public void service(ServletRequest
request ServletResponse ) throw
responses
throws ServletException, IOException.
  { // Servlet Task
}
public void destroy()
{
// finalization code
}
}.
```

Page No.
Date

(2) Difference Servlet & CGI.
(3) Difference Generic & HTTP servlet.
(4) Difference doGet & doPost.

} Darshan Notes Pg. 35/36 36/37.

Q. (2) Servlet Config.

- it is used to get configuration information from web.xml file.

- if the configuration information is modified from the web.xml file, we don't need to change the servlet.

Eg. String str = config.getInitParameter ("name")

(3) get object of ServletConfig :-

- getServletConfig() method of Servlet Interface returns the object of ServletConfig.

(4) Usage of ServletConfig :-

- if any specific content is modified from time to time. you can manage the web application easily without modifying the servlet through editing the value in web.xml.

Eg. ServletConfig config = getServletConfig();

Page No.
Date

④ Methods of ServletConfig Interface :-

(1) public string getInitParameter (String name):

- Returns the parameter value for the specified parameter name.

(2) public String ServletName() :-

- Returns the Name of servlet.

(3) public ServletContext getServletContext();

- Returns an object of ServletContext.

Q.③ ServletContext :-

- ServletContext is created by the web container at time of deploying the project.
- It can be used to get configuration information from web.xml.
- There is only one ServletContext object per web application.
- if any information is shared to many servlet, it is better to provide

it from the web.xml file using the <Context-param> element.

⑥ Advantages of ServletContext :-

- Easy to maintain.
- removes maintainance problem.

⑦ Usage of ServletContext :-

- There can be a lot usage of ServletContext object.

(1) the object of ServletContext provides an interface between the container & servlet.

(2) ServletContext can → get Configuration
(3) be used to → Information from web.xml
(4)

→ set, get or remove attribute from web.xml.

→ provide inter-application communicatio

⑧ Methods of ServletContext :-

(1) public String getInitParameter(String name):

- Returns the parameter value for the specified parameter name.

(2) public ~~Object~~ void setAttribute (String name, Object object):

- ~~Retu~~ sets the given object in the application scope.

(3) public void removeAttribute (String name);

- Removes the attribute with the given name from the servlet Context.

Q.4 Differentiate Servlet Config & Context.

Servlet Config	Servlet Context
1. ServletConfig object one per servlet class.	ServletContext is global to entire web application.
2. Object of ServletConfig will be created during Initialization process of the servlet.	Object of ServletContext will be created at the time of web application deployment.
3. Scope: As long As a servlet is executing, ServletConfig object will available, it will be destroyed once the servlet execution is completed.	Scope: As long As a web application is ex..... if it will ..... once the application is removed from server.

(4) we should give request explicitly, in order to create ServletConfig object for the first time.	ServletContext will be available even before giving the ~~object~~ first request.
(5) In web.xml - <init-param> tag will be appear under <servlet-class> tag.	In web.xml - <context-param> tag will be appear under <web-app> tag

Q.(5) What is Request Dispatcher? what is the difference between RequestDispatcher's forward() and include() method? Explain it in detail with program.

- the Request Dispatcher Interface provide the facility of dispatching the request to another resource.
- Resource can be HTML, Servlet or JSP.
- The interface can also be used to include the content of another resource
- It is one of the way of Servlet Collaboration
- the getRequestDispatcher() Method of ServletRequest interface returns the object of RequestDispatcher.

Page No.
Date

⊛ Syntax:

RequestDispatcher
getRequestDispatcher(String resource)

Ex. RequestDispatcher rd
= request.getRequestDispatcher
("servlet2");

rd.forward(request, response);
// method may be include / forward

⊛ there are two methods defined in the RequestDispatcher interface.

(1) forward()
(2) Include()

---

[1] forward():-

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

Syntax:- void forward
(ServletRequest request,
ServletResponse response)
throws ServletException, IOException

Page No.
Date

Ex : Forward() :-

// For Java servlet.

R.D rd = request.getR.D("Servlet2");
rd.Forward(request, response);

// For html page

" " ("/7.html");
" "

[2] Include() :-

- it includes the content of a resource (servlet, JSP page, or html file) in the response.

Syntax :- void include
(ServletRequest request,
ServletResponse response)
throws ServletException, IOException.

Ex : include() :-

// For java servlet

RD. rd = request.getR.D("servlet2");
rd.include(request, response);

// For html page

" " ("7.html");
" "

Page No.
Date

Q.(6) response.sendRedirect() :-

- The sendRedirect() method of HttpServletResponse interface can be be used to redirect response to another resource, it may be Servlet, jsp or html file.

Syntax :-

void sendRedirect (String location) throws IOException.

Eg.

response.sendRedirect ("http://www.roshanportfolio.netlify.app");
response.sendRedirect("7.html"); // relative path.

↑ send

(7) Differentiate forward() Redirect().
↑ Darshan pg. 50 Notes

Q.(7) What is session? Why we require session? Explain Session management techniques.
or,
List out Various session managment techniques. or different ways to manage the session.

- A session refers to the entire interaction between a client and a server from the time of client's first request, which generally begins the session, to the time last request / response!!

(*) why we require session?
/use of session :-

- HTTP is a 'stateless' protocol which means each time a client retrieves a web page, the client opens a separate connection to the web server & the server automatically does not keep any record of previous client request.

- Session is required to keep track of users and their information.

(*) List different ways to manage the session. / session management techniques.

- session Management is a mechanism used by the web containes to store information for a particular user.

- There are 4 different techniques for session management.

(1) Hidden form field (3) Cookies
(2) URL Rewriting (4) HttpSession.

(1) Hidden Form Field :-

- Hidden form field, a hidden (invisible) textfield is used for maintaining the state of an user.

- In such case, we store the information in the hidden field and get it from another servlet.

Eg. <input type = "hidden" name = "session-id" value = "054">

(*) Applications :-

- Widely used in comment form of a website.

(*) Advantages :-

- Easy to implement
- it will always work whether cookie is established or not.

(*) Disadvantages :-

- it is maintained at server side.
- Security
- Only textual information can be used.
- Extra form submission is required on each pages.

Page No.
Date

(2) URL Rewriting :-

- In URL Rewriting, a token or identifier is appended to the URL of the next servlet or the next resource.

- we can send parameter name/value pairs using following format:

URL? Name1 = Value1 & name2 = value2 &....

- Here, A name & value is seperated using an equal (=) sign and name/value pair is seperated from another parameter using the ampersand (&).

- When the user click hyperlink, the parameter name/value pair will be passed to the server.

- From servlet we can use getParameter() method to maintain a parameter value.

Advantages :-

- Extra submission form is not required on each pages.
- Always work whether cookies is establish or not.

disadvantages :-

- work only with links.
- size URL size constraint
- security.
- Store only textual information

(3) Cookie :-

Q. What is Cookie? What does the Cookie contains? Explain methods of Cookie class.

- A Cookie is a small piece of information that is persisted between the multiple client request.

OR

- Cookies are the textual information that is stored in key-value pair format to the client's browser during multiple requests.

- A Cookie has a
  - Name
  - single value
  - optional attributes such as

i. comment
ii. path
iii. domain qualifiers.
iv. a maximum age
v. version number etc.

⊛ Types of Cookies :-

- There are two types of cookies in servlet.

(1) Non-Persistent/ session Cookie
(2) Persistent Cookie.

(1) Session Cookie :-

- it is valid for single session only. it is removed each time when user close the browser.

(2) Persistent Cookie :-

- it is valid for multiple session. it is not removed each time when user closes the browser. it is removed only if user logout or signout.

(3) Cookie class (javax.servlet.http.Cookie)

- This class provides the functionality of using cookies.

- it provides a lots of useful methods for cookies.

(4) Constructor :-

Cookie (String name, String value):

- constructs with a cookie with a specified name & value.

Ex. Cookie c = new Cookie ("session-id", "054");
// creating cookie object.

⑧ Methods of Cookie class :-

(1) Void setMaxAge (int expiry):

- sets the maximum age in seconds for this cookie.

(2) int getMaxAge ():

- gets the maximum age in seconds for this cookie.

Note :- By default, -1 is return, which indicates that the cookies will persist until browser shutdown.

(3) String getName():

- Returns the name of cookie. The name cannot be changed after creation.

(4) Void setValue (String newValue):

- Assigns a new value to this cookie.

(5) String getValue ():

- gets the current value for this Cookie.

(6) void addCookie (Cookie cookieObj):

- Method of HttpServletResponse interface is used to add Cookie in response object.

(7) Cookie[] getCookies():

- Returns an array containing all of the Cookies objects the client sent with this request. This method returns null if no cookies were sent.

Advantages :-

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantages :-

- it will not work if cookie is disabled from the client. browser.
- Only textual information can be sent set in Cookie object.

How to Create Cookie?

```
// creating cookie object.
Cookie c = new Cookie ("session-id"
                       "054");
```

// adding cookie in the response from server to client.

response.addCookie(c);

④ How to retrieve Cookies?

Cookie c[] = request.getCookies();

For(int i=0; i<c.length; i++)
{
out.print(c[i].getName()
+ " " + c[i].getValue());
// printing name & value of cookie
}

⑤ How to delete Cookie?

1. Read an already existing cookie & store it in Cookie object.
2. set cookie age as zero using setMaxAge() method to delete an existing cookie
3. Add this Cookie back into response header.

// deleting value of cookie
Cookie c = new Cookie("user", "");

// changing the maximum age to 0 seconds.
c.setMaxAge();

// adding cookie in the response:
response.addCookie(c);

(4) Session Management using HttpSession :-

- HttpSession Interface (javax.servlet.http.HttpSession) provides a way to identify a user across more than one page request.

- The container creates a session id for each user.

- The container uses this id identify the particular user.

- An Object of HttpSession can be used to perform two tasks:

1. Bind objects
2. View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

- HttpSession object is used to store entire session with a specific client.

- We can store, retrieve & remove attribute from HttpSession object.

- Any servlet can have Access to HttpSession object throughout the getSession() method of the HttpServlet object.

- The servlet Container uses this interface to create a session between an HTTP client & an HTTP server.

- In this technique create a session object at server side for each client.

- Session is available until the session time out, until the client logout.

- $ the default session time is 30 minutes and can configure explicit session time in web.xml file.

⊛ Working of HttpSession :-

(1) On client's first request, the web Container generates a unique session ID and gives it back to the client with response.

(2) The client sends back the session ID with each request. ~~Make~~ Making it easier for the web container to identify where the request is coming from.

(3) The web container uses this ID, finds the matching session with the ID & associates the session with the request.

(*) Methods of HttpServletRequest interface :-

- Darshan Notes : pg. no - 62.

(*) How to create the session?

```
HttpSession hs = request.getSession();
hs.setAttribute("s-id", "diet54");
```

(*) How to retrieve a session?

```
HttpSession hs = request.getSession(false);
String n = (String)hs.getAttribute("s-id");
```

(*) How to invalidate a session?

```
hs.invalidate();
```

Q. (*) What is filter? what is its use? List different filter interfaces with their important methods. List applications of filter.

(*) Explain the configuration of filter using deployment descriptor.

- A Fileter is an object that invoked at the preprocessing & post processing of a request. Java servlet Filete is used to intercept request and do some pre-processing and can be used to intercept response and do post-processing before sending to client in web-applicatio

Use of Filters :-

- Recording all incoming requests
- Logs IP address of the computers from which the requests originate
- conversion
- Data compression
- Encryption and Decryption
- Input Validation etc.

Filter Interfaces :-

- the javax.servlet. package contains the three interfaces of Filter API.

(1) Filter :-

- For creating any filter, you must implement the filter interface.
- Filter interface provides the life cycle methods for a filter.

Page No.
Date

(1) Method of Filter Interface :-

(1) void init(FilterConfig config):

- init() method is invoked only once. it is used to initialize the filter.

(2) void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain):

- doFilter() method is invoked every time when user request to any resource, to which the filter mapped. it is used to perform filtering tasks.

(3) void destroy():

- This is invoked only once when filter is taken out of the service.

(2) FilterChain :-

- The object of FilterChain is responsible to invoke the next filter or resources in the chain.

- The object is passed in the doFilter method of filter interface.

Page No.
Date

⊛ Method of FilterChain Interface :-

(1) void doFilter(HttpServletRequest request, HttpServletResponse response):

- it passes the control to the next filter or resource.

(3) FilterConfig :-

- FilterConfig is created by the web container.

- This object can be used to get the configuration information from the web.xml file.

⊛ Methods of FilterConfig Interface :-

(1) void init (FilterConfig config):

- init() method is invoked only once it is used to initialize the filter.

(2) String getInitParameter (String parameterName):

- Returns the parameter value for the specified parameterName.

Page No.
Date

⑧ Applications of filter :-

1. Authentication - Blocking requests based on user Identity.
2. Logging and auditing - Tracking users of a web-application.
3. Image Conversion - Scaling maps, and so on.
4. Data Compression - making downloads smaller.
5. Localization - Targeting the requests and response to a particular locale.

⑧ Advantages of filter :-

- Filter is pluggable.
- one filter don't have dependency onto another resources.
- Less maintainance cost.

Q.⑨ SessionTimeout :-

:- The SessionTimeout in a web application can be configured in two ways:

(1) Timeout in the deployment descriptor (web.xml).
(2) Timeout with setMaxInactiveInterval()

(1) Timeout in the deployment descriptor (web.xml):

- <web-app>
  <session-config>
  <session-timeout> 30
  </session-timeout>
  </session-config>
  </web-app>

- Note that the value of the timeout is set in minutes, not in seconds.

(2) Timeout with setMaxInactiveInterval():

- The timeout of the current session only can be specified programmatically via the API of the javax.servlet.http.HttpSession.

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(10*60);
// time specified in seconds.
```

Q.(8) State & explain Types of Servlet Events:

- Dasshen Notes
  - pg.no - 72/73.

Q.(11) Methods

(1) HttpSessio

- Returns with H does no

(2) HttpSessio

-- Returns with H current returns

(3) HttpSession.

-

(4) Long

- Returns uses c

(5) Long g

- Returns request as the

Q.(11) Methods of HttpServletRequest Interface:

(1) HttpSession getSession():-

- Returns the current session associated with this request, or if the request does not have a session, create one.

(2) 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.

HttpSession.

(3) String getId():-

- Returns a string containing the unique identifier value.

(4) Long getCreationTime():-

- Returns the time when this session was created, measured in milliseconds.

(5) Long getLastAccessedTime():-

- Returns the last time the client sent a request associated with this session, as the number of milliseconds.

(6) void Invalidate() :-

- Invalidates this session then unbinds any objects bound on it.

Q.(12) Methods of HttpServletResponse (I) * — Interface

(1) String getCharacterEncoding() :-

- it returns the name of the MIME charset that was used in the body of the client response.

(2) String getContentType() :-

- It returns the response content type. eg. text, html, etc.

(3) PrintWriter getWriter() :-

- the PrintWriter object is used to transmit character text to the client.

(4) ServletOutputStream getOutputStream() :-

- this method returns a ServletOutputStream that may be used to write binary data to the response.

⊛ Chapter : ⑦ : Spring MVC ⊛

⑦.① Architecture of Spring MVC Framework

- Spring's web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.

- Spring's DispatcherServlet is completely integrated with Spring IOC container and allows us to use every other feature of Spring.

⊛ Request process lifecycle of Spring 3.0 MVC :-

(1)- The client sends a request to web container in the form of http request.

(2)- This incoming request is ~~inter~~ intercepted by Front Controller (DispatcherServlet) and it will then tries to find out appropiate Handler mappings.

(3)- with the help of Handler mappings, the DispatcherServlet will dispatch the request to appropiate controller.

(4) - The controller tries to process the request and returns the Model and View object in form of Model And View instance to the Front Controller.

(5) - The Front controller then tries to resolve the view by consulting the View Resolver object. The selected view is then rendered back to client.

Q. (2) What is Spring Web MVC Framework? List its key Features.

- Spring links objects together instead of the objects linking themselves together.
- Spring object linking is defined in XML files, allowing easy changes for different application configurations thus working as a plug in architecture.
- In an MVC architecture your controllers handle all requests.
- Spring uses a "DispatcherServlet" defined in the web.xml file to analyze a request URL pattern and then pass control to the correct Controller by using a URL mapping defined in a "spring bean" XML file. All frameworks integrate well with spring.

- Consistent Configuration, open plug-in architecture.
- Integrates well with different O/R Mapping Frameworks like Hibernate
- Easier to test applications with.
- Less complicated than other Frameworks.
- Active user community.
- Spring is well organized and seems easier to learn comparatively.
- Spring also supports JDBC Framework that makes it easier to create JDBC Apps.

② Advantage of Spring MVC Framework

(1) Predefined Templates
(2) Loose Coupling
(3) Easy to test
(4) Lightweight
(5) Fast Development
(6) Declarative Support
(7) Hibernate and JDBC Support
(8) MVC Architecture & JavaBean Support.

Page No.
Date

(8) Features of Spring MVC Framework :-

(1.) Inversion of Control (IoC) Container.

- it is used to provide object reference to class during runtime.

(2) Data Access Framework.

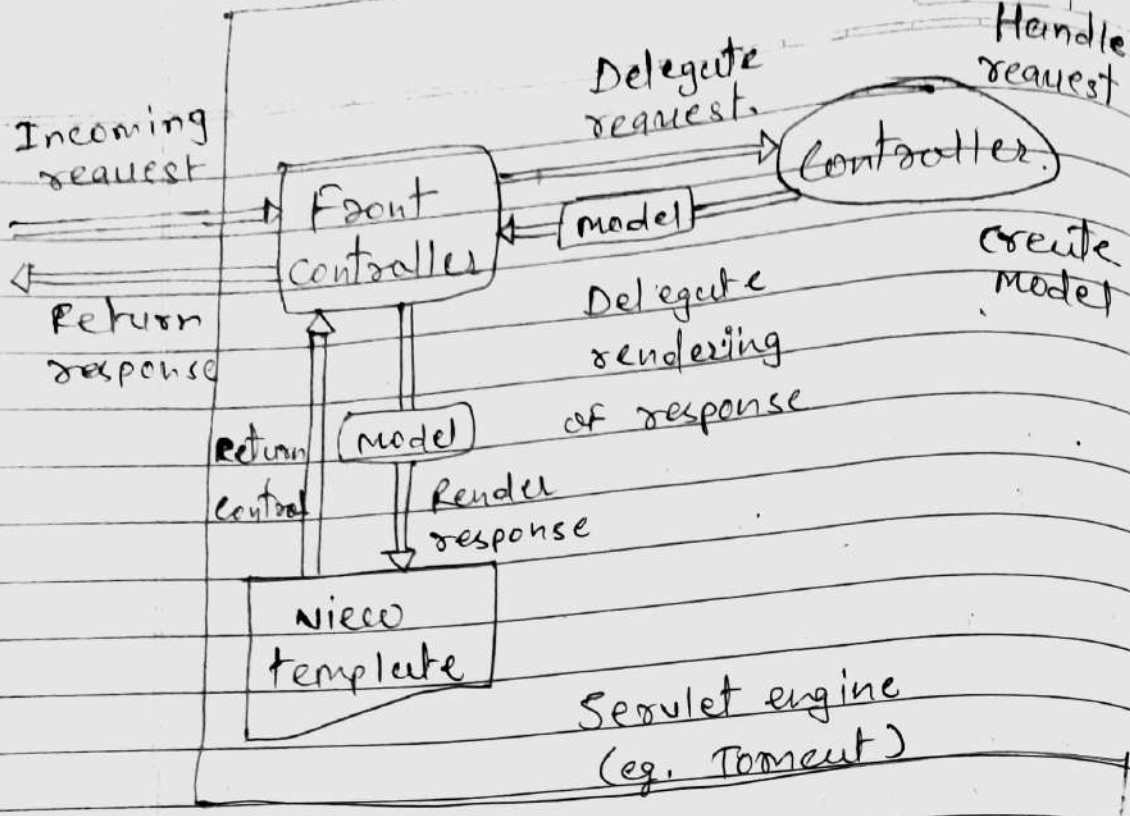
- it enables developers to easily write code to access the persistent data through the application.

(3) Transaction Management

- it enables developers to model a wide range of transaction by providing Java Transaction API (JTA).

(4) Spring Web Services.

- It provides powerful mapping for transmitting incoming XML request to any object.



① Spring MVC Architecture ②

② Features of Spring 3.0 :-

- Sprive 3.0 Framework Jeeve 5. It provides annotation based configuration support. Java 5 features such as generic, annotations, Varags etc can be used in Spring.

- A new expression language Spring Expression language SpEL is being introduced. The Spring Expression language can be used while defining the XML and Annotation based bean definition.

- Spring 3.0 Framework supports RESI web services.

- Data Formatting can never be so easy. Spring 3.0 supports annotation based formatting. We can now use the @DateTimeFormat (iso = ISO.DATE) and @NumberFormat (style = Style.CURRENCY) annotations to convert the date and currency formats.

- Spring 3.0 has started support to JPA 2.0.

Q. ③ Dependency Injection :-

- Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage & test the application.

- Dependency Injection makes our program code loosely coupled.

- DI is a concept of Injecting an object into a class rather than explicitly creating object in a class, since IoC container injects object into class during runtime.

Ex: Standard code without Dependency Injection.

```
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor() {
        spellChecker = new SpellChecker();
    }
}
```

Ex: code with Dependency Injection

```
public class TextEditor {
    private SpellChecker spellChecker;
    public TextEditor(SpellChecker
                      spellChecker) {
        this.spellChecker = spellChecker;
    }
}
```

- Here, the TextEditor should not worry about SpellChecker implementation.
- The SpellChecker will be implemented independently and will provided to the TextEditor at the time of TextEditor instantiation.

- This entire procedure is controlled by the spring Framework.

Q. (4) IoC container :-

- The spring container is at the core of the Spring Framework.

- the containes will create the objects, wire them together, configure them, and manage thier complete life cycle from creation till destruction.

- The spring container uses DI to manage the components that make up an application.

- The main tasks performed by IoC container are:

(1) to Instantiate the application class
(2) to configure the object
(3) to assemble the dependencies between the objects.

- There are two types of IoC containers:

(1) BeanFactory
(2) Application Context

Q. ① Spring Bean life cycle.

- The life cycle of spring Bean is easy to understand.

- When a bean is initiated, it may be required to perform some initialization to get it into a useable state.

- Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.

- Though, there is lists of the activities that take place behind the scenes between the time of bean instantiation and its destruction, but this chapter will discuss only two important bean lifecycle callback methods which are required at the time of bean ~~first~~ initialization and its destruction.

- To define setup and teardown for a bean, we simply declare the <bean> with init-method and/or destroy-method parameters.

- The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation.

- similarly, destroy-method specifies a method that is called just before a bean is removed from the container.

④ HelloWorld.java.

```
public class HelloWorld {

   private String message;

   public void setMessage(String message)
   {
      this.message = message;
   }

   public void getMessage() {
      System.out.println("your message" + message);
   }

   public void init() {
      System.out.println("Bean is going
                          through init.");
   }

   public void destroy() {
      System.out.println("Bean will destroy now.");
   }
}
```