* What is JDBC ? Explain its architecture.

→ JDBC is a Java API used to connect and interact with databases.

- It provides a set of classes and interfaces for database operations.
- JDBC enables Java applications to execute SQL queries and retrieve data.
- It acts as a bridge between Java programs and database servers.
- Supports various relational database like MySQL, Oracle & Postgre SQL.
- JDBC uses SQL for performing database operations like CRUD.
- It includes drivers for comm. with specific databases.
- JDBC is part of Java Standard Edition (Java SE).

→ JDBC Architecture :-

- The architecture of JDBC can be understood in the following points:

vision

1. JDBC API:
- This is the interface provided by Java for database access. It contains classes and methods to connect, send queries & retrieve results from a database. The API enables the application to interact with various databases using standard SQL.

2. JDBC Driver Manager:
- It manages the communication between Java applications and database drivers. The DriverManager class loads the appropriate driver and establishes the connection to the database.

3. JDBC Driver:
- This is the database-specific implementation of the JDBC interface. It translates Java calls into database-specific operations. There are four types of JDBC drivers:

Type-1: JDBC-ODBC Bridge Driver

Type-2: Native-API Driver

Type-3: Network Protocol Driver

Type-4: Thin Driver (Pure Java Driver)

4. Connection Interface:
- This interface represents the connection with the database. It provides methods to create statements, manage transactions, handle connection properties.

5. Statement Interface:
- This interface is used to execute SQL queries against the database. The statement, PreparedStatement and CallableStatement interfaces are its variations for executing queries, executing precompiled queries and calling stored procedures -> respec.

6. ResultSet Interface:
- It is used to store and manipulate the results returned from executing SQL queries. It provides methods to iterate through the queries, retrieve data & update rows.

7. JDBC URL:
- The database connection URL specifies the database server, port and other connection details. It helps JDBC identify the correct database to connect to.

vision

8. SQL Exception Handling:
- JDBC provides the SQL Exception class to handle database-related errors, like connection failure or incorrect SQL syntax.

* Explain common Components of JDBC.
- Common components:

1. Driver Manager:
- Manages a list of database drivers
- Establishes a common connection between Java applications and the database.
- Automatically selects the appropriate driver based on the database URL.

2. Driver:
- An interface for database-specific drivers that communicate with the database.
- Converts JDBC calls into database-specific commands.
- There are 4 types of drivers

3. Connection:
- Represents the connection to the database
- Provides methods to create Statement & manage transactions.
- Manages connection attributes

4. Statement :
- Executes SQL queries & return results
- Three types:
  - Statement : For simple SQL queries
  - PreparedStatement : Precompiled SQL queries for better performance
  - CallableStatement : Used to execute stored procedures.

5. ResultSet :
- Stores the result of SQL queries
- Allows iteration over results row by row
- Provides methods to retrieve data in various types

6. SQLException :
- Handles database-related exceptions.
- Provides details about the error, including error codes & messages.

vision

* Explain two tier, three tier Architecture for Data Access.

→ Two-Tier ~~Database Design~~ and Three-Tier Architecture are common data access architectures that define how data flow between the client application, server and additional intermediary layer.

→ Two-Tier Architecture :-

- In Two-Tier Architecture, the client interacts directly with the database. There are two main layers:

1. Client (Application layer):

- This is the front end where users interact with the application.
- The client application contains the user interface and handles requests, sending them directly to the database.
- Desktop applications using JDBC to connect directly to the database.

2. Server (Database layer):

- The back end, where the database is located, processes and stores data.

- The database server receives SQL queries from the client, processes them and returns the results.

→ Use cases :-

- Suitable for standalone applications, small-scale systems or applications with a small number of users, such as personal finance software or desktop inventory systems.

→ Three-Tier Architecture :-

- In a Three-Tier Architecture, an additional middle layer is introduced between the client and the database server. This intermediary layer, often called the Application Server or Business logic layer, handles processing, business rules and communication between the client and database.

→ The three main layers are:

1. Client (Presentation layer):
- The front end where the user interface resides
- Users interact with the application to send requests
- The client sends requests to the middle layer instead of directly accessing the database.

2. Application Server (Business logic-layer):

- This intermediary layer receives requests from the client and processes them based on defined business rules.
- It can validate inputs, execute business logic, perform calculations and handle transaction mgmt.
- The app. server then forward the requests to the database, receives the responses, processes them as needed & sends back to the client.

3. Database Server (Data layer):

- The back-end that manages data storage, retrieval & manipulation.
- Processes SQL queries / data requests

→ Use cases:-

- Common in web app., enterprise systems, any app. needing robust security, scalability & support for multiple clients.

* Explain JDBC Driver types.

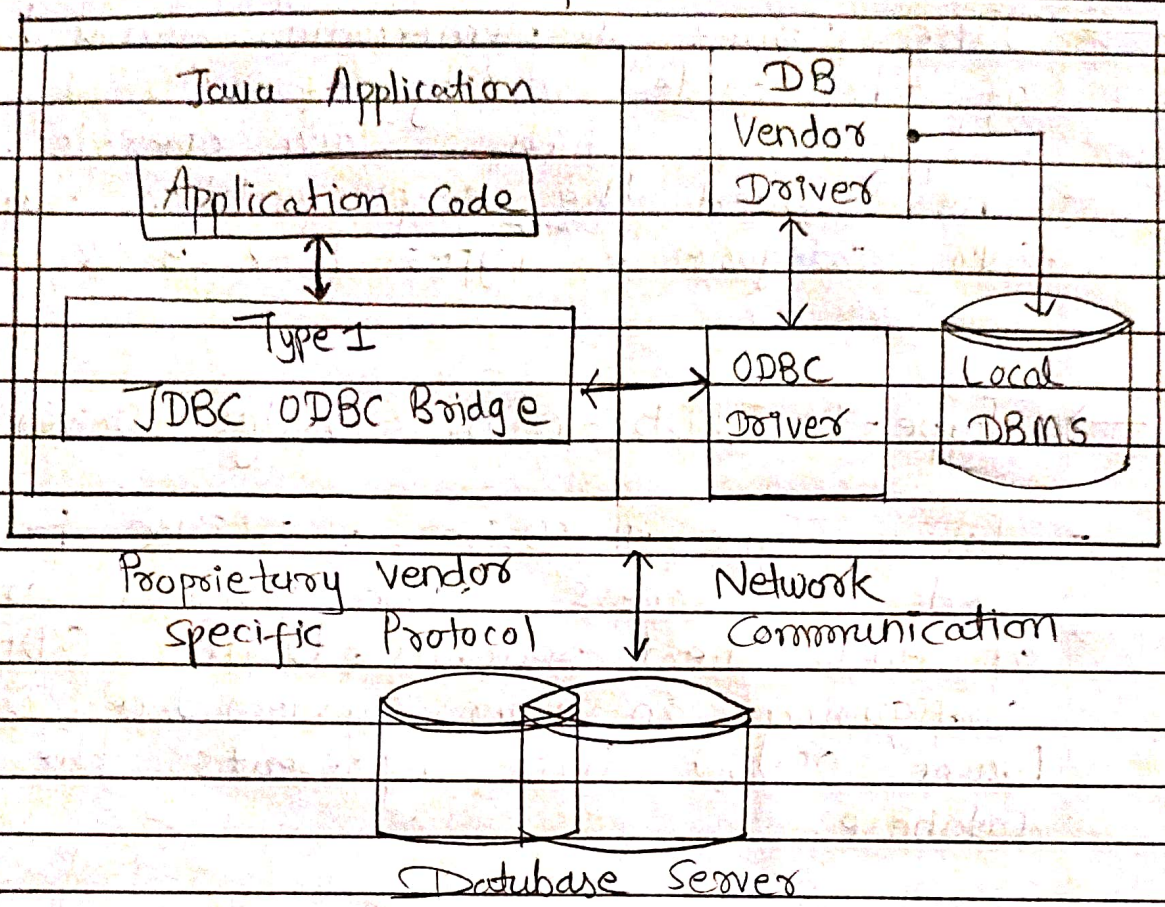
→ JDBC Driver implementations vary because of the wide variety of OS & hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3 & 4.

→ Type-1: JDBC-ODBC Bridge Driver

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

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

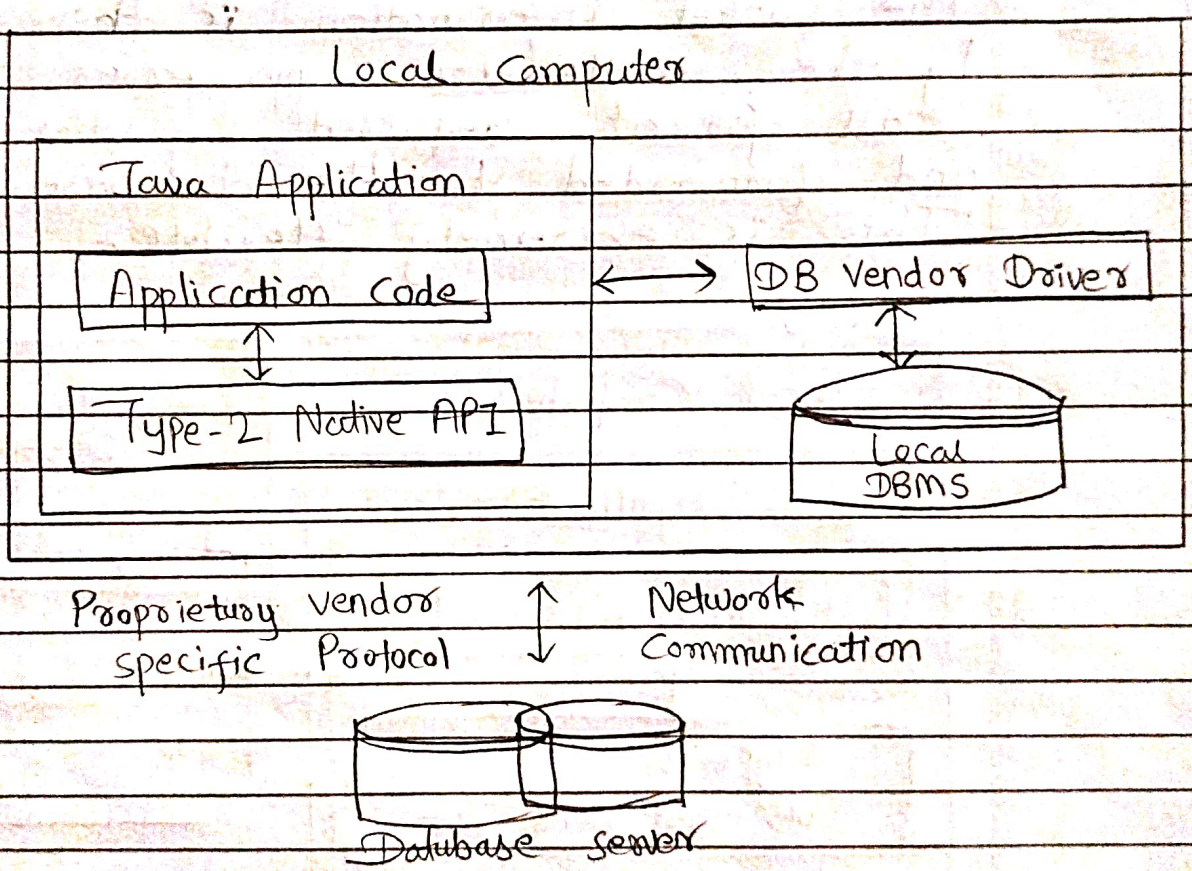
- The JDBC-ODBC Bridge that comes with JDK 1.2 is good example of this kind of driver.



→ Type-2: JDBC Native API

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

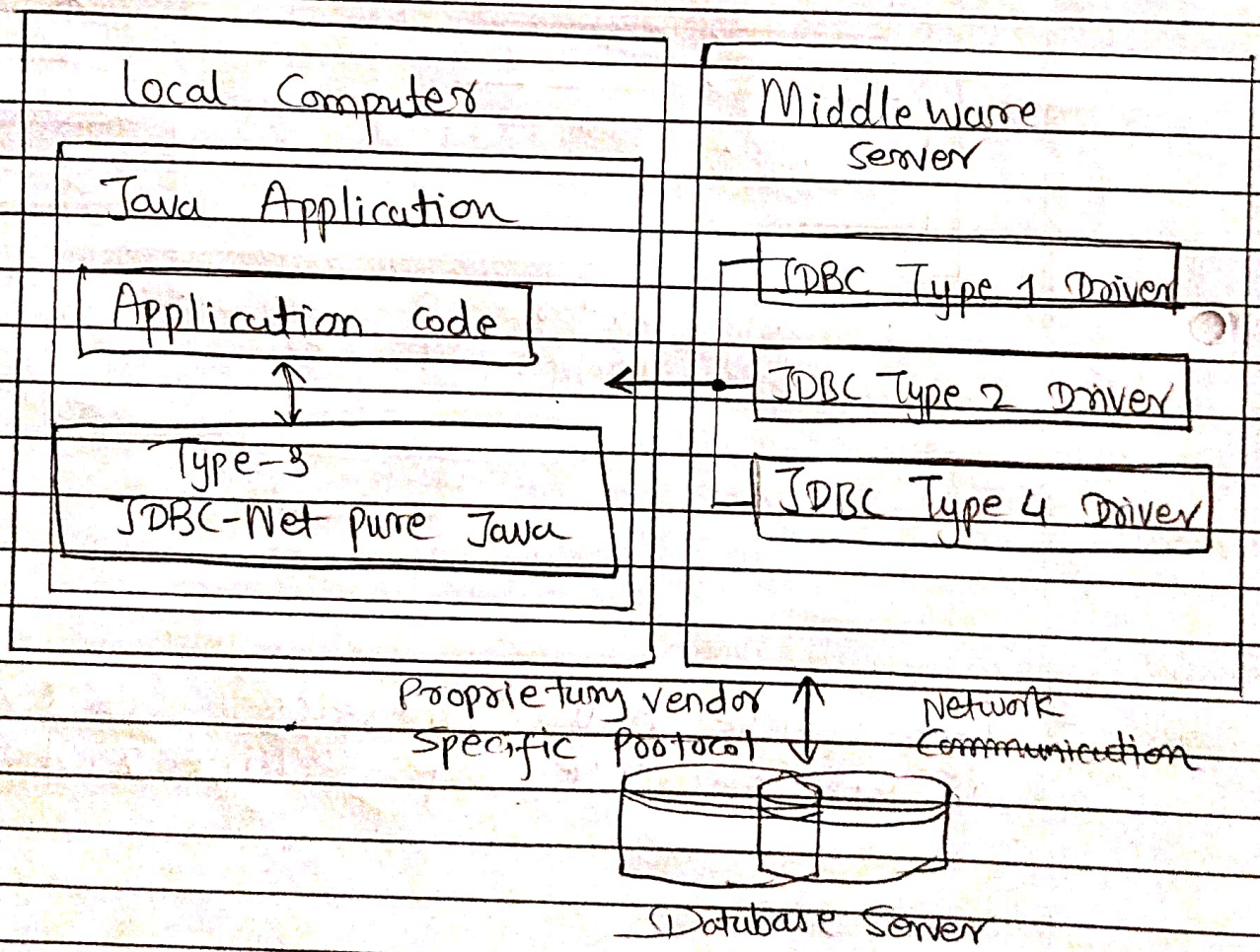
- If we change the Database we have to change the native API as it is specific to a database and they are mostly obsolete now but you may realize some increase with a Type 2 driver, because it eliminates ODBC's overhead.



- The oracle call Interface (OCI) driver is an example of Type 2 driver.

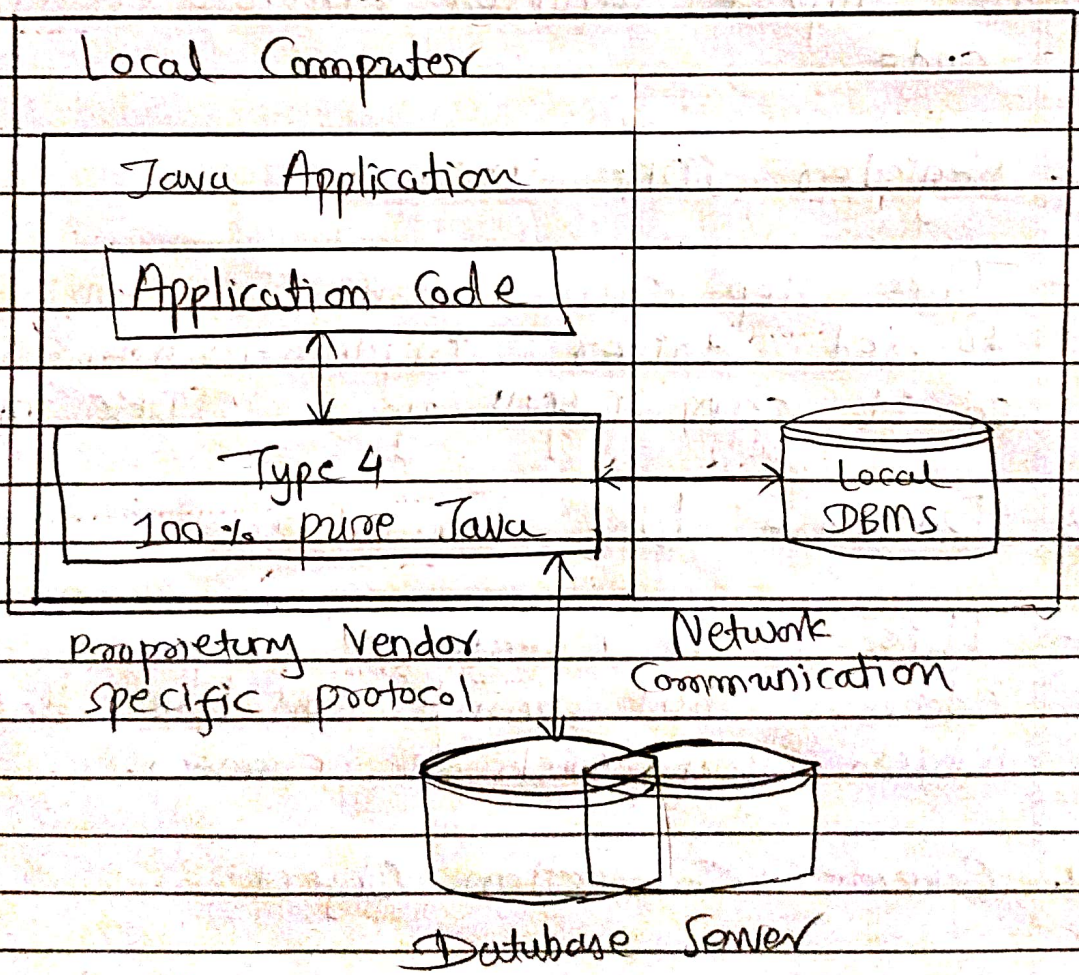
→ Type-3: JDBC - Net pure Java:

- In type-3 driver, a three-tier approach is used to access database. The JDBC clients use standard network sockets to communicate with a middleware application server.
- The socket information is then translated by the middleware app. server into the call format required by the DBMS and forwarded to the database server. This is extremely flexible.



→ Type - 4: 100 % pure Java:

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



* What do you mean by JDBC Statement, Callable Statement and Prepared Statement Interface.

→ The JDBC Statement, CallableStatement and PreparedStatement interfaces define the methods and properties that enables you to send SQL or PL/SQL commands and receive data from your database.

→ Statement

- Use for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

→ Prepared Statement

- Use when you plan to use the SQL statements many times. The prepared-Statement interface accepts input parameters at runtime.

→ Callable Statement

- Use when you want to access database

stored procedures. The callable statement interface can also accept runtime input parameters.

* What is the life cycle of a Servlet ?

→ The lifecycle of a servlet gives the states of servlet. There are three states of servlet - new, ready and end. The states of servlets change with the help of the calling of servlets method.

→ Servlet Interface:

- It provides a common functionality in all the servlets. The interface defines the methods that all the servlets must implements.

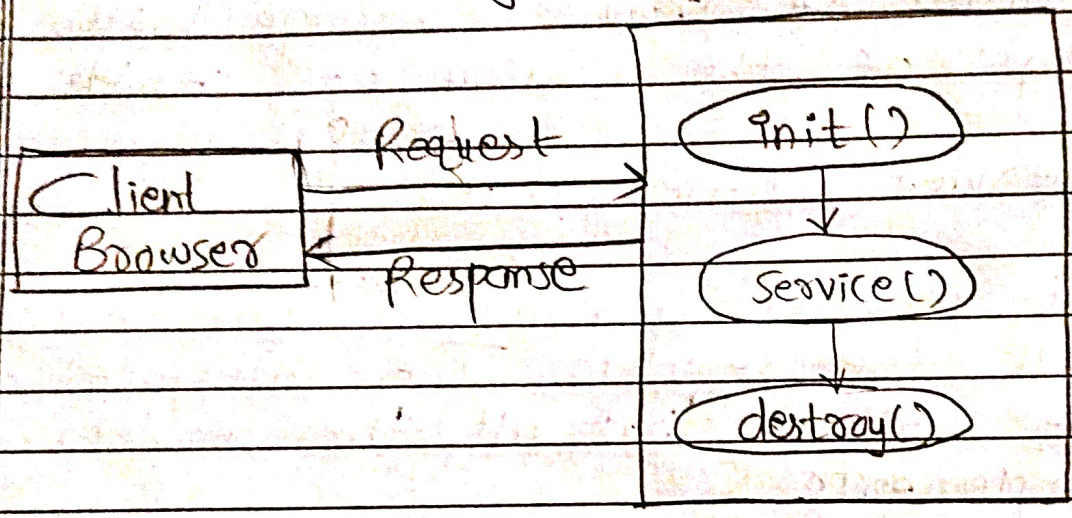
- HTTP Servlet & GenericServlet implement Servlet interface directly or indirectly.

- A servlet is loaded only once in the memory and is initialized in the init() method.

- After the servlet initialized, it starts accepting a request from the client and processes them through the service() method until it is shut down by the destroy() method.

- The service() method is executed for every incoming request. The lifecycle of a servlet is depicted as follows:

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



* What are some of the advantages of servlet?

→ Advantages:-

1. Platform Independence:
- Servlets are written in Java, making them platform-independent. They can run on any server that supports the Java servlet API, making them a good choice for applications that need to run on multiple OS.

2. Performance:
- Servlets provides high performance since

they are executed within the server process itself. Unlike CGI, which spawns a new process for each request, Servlets run as a single instance that handles multiple requests, which reduces resource usage and increases response time.

3. Scalability:
- Servlets can efficiently handle multiple requests by creating a single instance that serves multiple clients using threads.

4. Robustness:
- Java provides extensive error handling & memory mgmt features, which make Servlets reliable.

5. Ease of Integration:
- Servlets can easily interact with Java-based APIs & libraries, enabling seamless integration with other java technologies like JDBC & JavaMail

6. Session Management:
- Servlet support HTTP session mgmt, allowing the server to track user session across multiple requests.

7. Security:
- Servers run on the server side & better suited to handle secure requests that client-side tech..

* Explain the difference between the Get and Post Method.

Aspect	GET	POST
Data Transmission	Data is appended to the URL as a query string	Data is sent through a separate socket connection.
Data size Limit	limited to 240 characters	Can send a large amount of data without strict limitations
Visibility	Query String is visible in the browser's address bar, posing a potential security risk.	Data is not visible in the browsers, providing better security
Speed	Faster, data is part of URL	Slower, data is sent in separate packets.
Security	less secure since the data is part of the URL & can be cached / logged	More secure, as the data is not appended to the URL & is less exposed
Usage	Commonly used for data retrieval, such as search queries.	Used for data submission such as form submission.

* What are servlet filters?

→ Servlet filters are components in Java Web app. that allow developers to intercept & manipulate incoming request & outgoing response in a web application.

→ Key characteristics of servlet filters:

1. Pre-processing & Post-processing:
- Filters can intercept requests before they reach a servlet & can modify responses after a servlet has processed the request

2. Reusability:
- Filters are designed to be reusable and can be applied to multiple servlets or JSP pages without modifying the servlet code itself.

3. Modularity:
- They help separate cross-cutting concerns from business logic, promoting a clean, modular design.

4. Chain of Filters:
- Filters can be configured in a chain where multiple filters are applied in a specified order.

* What is session Tracking?

→ A session is a group of activities performed by a user while accessing a particular website. The process of keeping track of the settings across the sessions is called session Tracking.

- It can also be used to keep track of the user's preferences.

* How does cookies work in servlets?

→ Cookies are small text files used by a web server to keep track of the users. A cookie has values in the form of key-value pairs.

- They are created by the server & sent to the client with the HTTP response headers. The client saves the cookies in the local hard disk and sends them along with the HTTP request headers to the server. If a cookie with the same name already exists, then the key is overwritten with the new value. A server can send one or more cookies to the client.

* What is MIME type ?

→ The contentType attribute define the MIME type of the response.

- MIME is a standard that specifies how the messages need to be formatted when exchanged between the different systems.

- A MIME message can consume diff. types of data like text, images, audio, video.

* What is the purpose of Request Dispatcher Interface ?

→ The Request Dispatcher interface is used to forward or delegate a request from a servlet to other resources, such as a servlet, an HTML file, or a JSP page. In this case, the source servlet does some processing & delegates the request to another servlet.

- A servlet Context is a directory in which the servlets are deployed in the Webserver. Servlets that execute in the same server belong to the same servlet context. However, few web servers also enable the creation of more that one servlet context.

* Difference between forward() method & sendRedirect() method?

Feature	Forward()	SendRedirect()
Method location	part of the RequestDispatcher interface	Part of HTTPServletResponse interface
Working Mechanism	Works on the server-side; forwards the request within the server	Works on the client-side; issues a new request from a client
Request & Response	Forwards the same request & response to the target resource	Sends a new request to the target resource
Client Awareness	The client is unaware of the forwarding & the URL in the browser remains the same	The client is aware of the redirection, the URL changes to the new target
Usage scope	Limited to resources within the same web app.	can redirect to resources inside / outside the app.
Speed	Faster as it occurs within the server	Slower due to additional client-server round trip
When to use	Suitable for internal resources like JSP, servlets etc	Suitable for external resources, or when a complete URL change is needed

vision

* What is JSP ? Explain its architecture ?

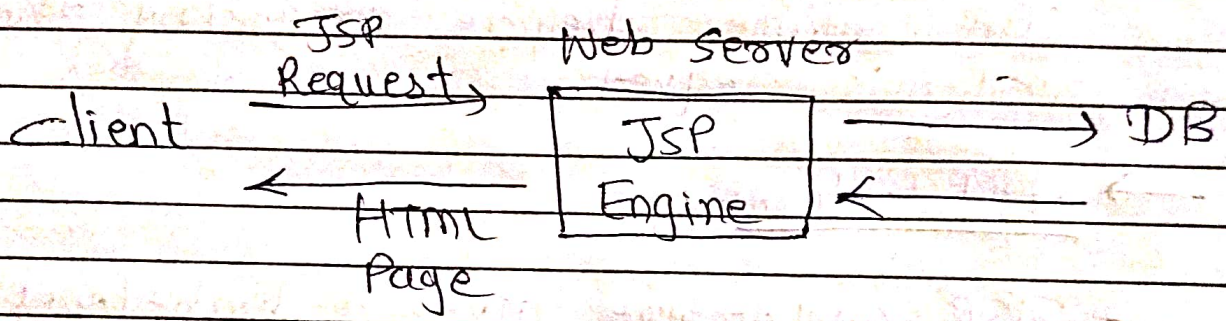
→ JSP stands for Java Server Pages

- It is a server side technology
- It is used for creating web app. & dynamic web content.
- In this, JSP tags are used to insert JAVA code into HTML/ XML pages or both
- JSP is first converted into servlet by JSP container before processing the client's request.

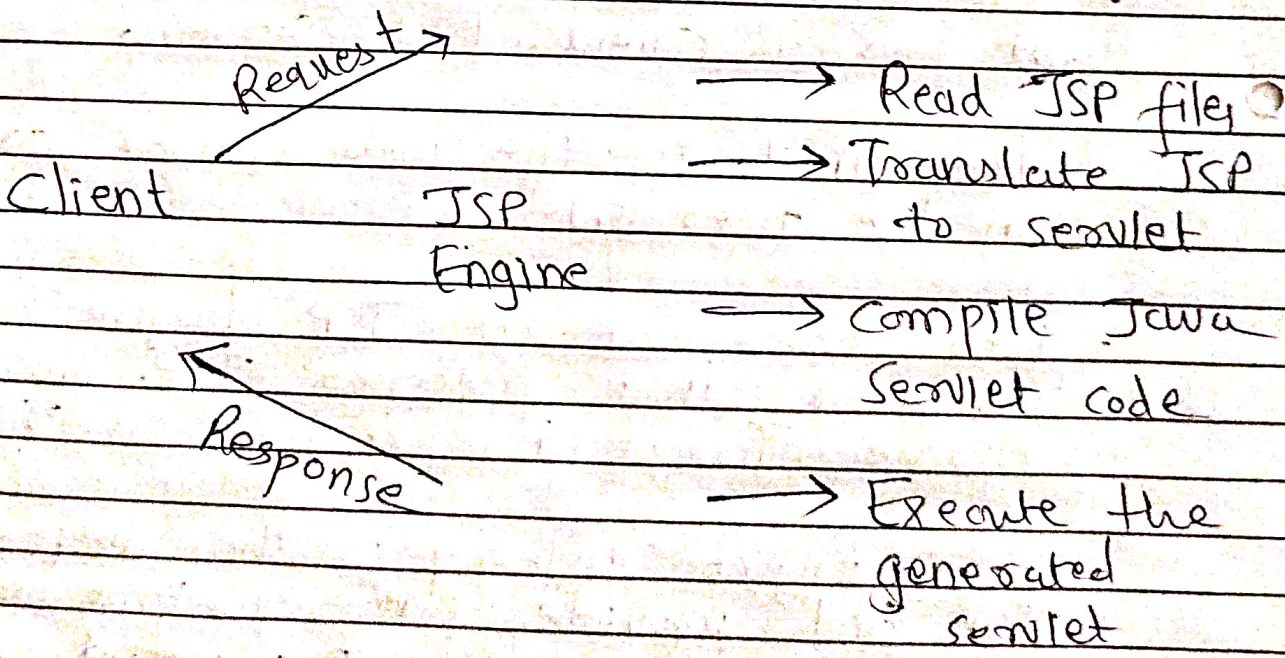
→ Architecture :-

- JSP architecture gives a high-level view of the working of JSP
- JSP architecture is a 3 tier architecture. It has a Client, Web Server, Database.
- The client is the web browser or app. on the user side.
- Web server uses a JSP Engine i.e; a container that processes JSP. For ex, Apache Tomcat has a built-in JSP Engine
- JSP Engine intercepts the request for JSP and provides the runtime environment for the understanding and processing of JSP files

- It reads, parses, build Java Servlet, compiles and Executes Java code & returns the HTML page to the client
- The Webserver has access to the Database. The diagram on next slide shows the architecture of JSP.



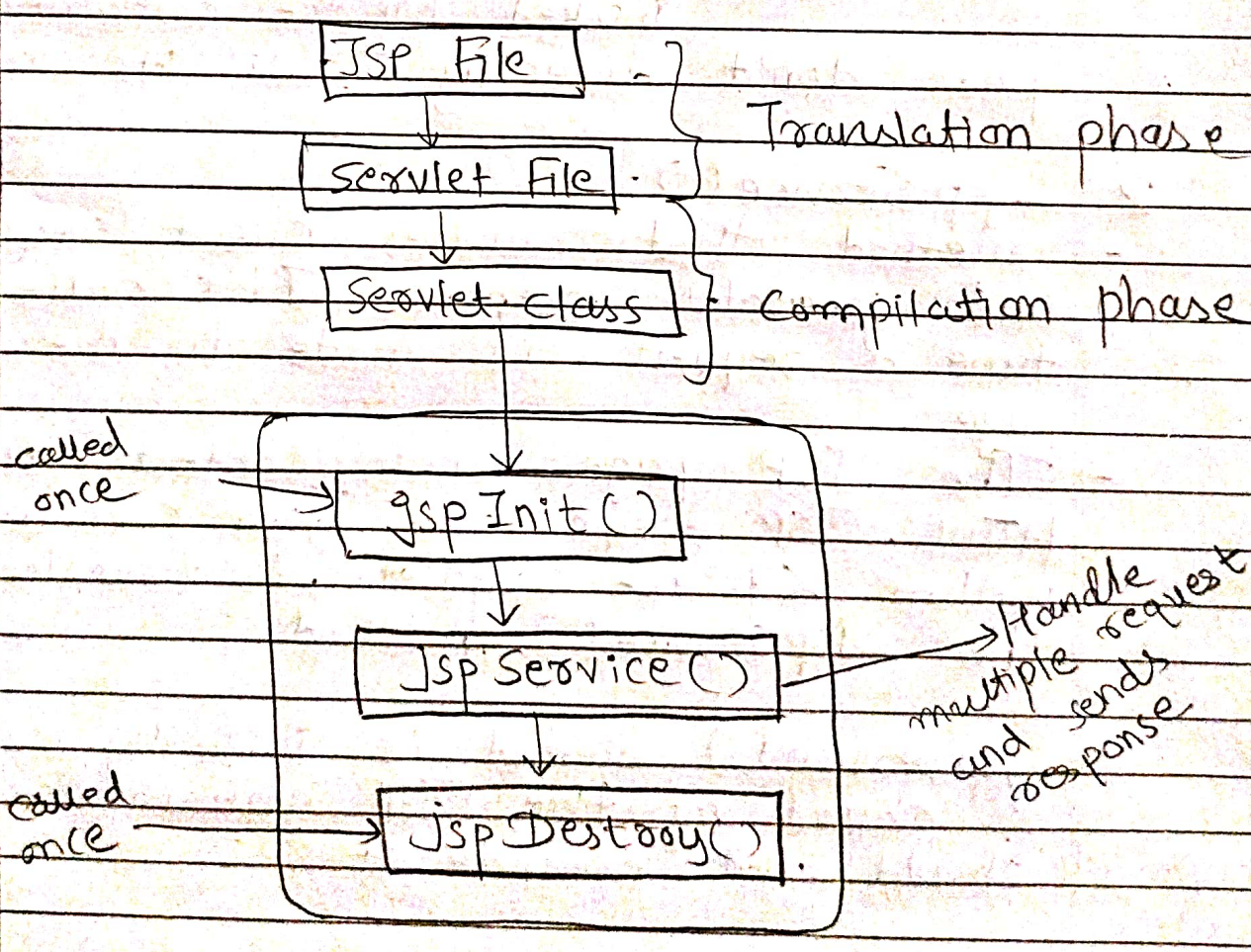
→ JSP processing :-



* What are the life-cycle methods for JSP ?

→ A Java Server page life cycle is defined as the process that started with its creation which later translated to servlet and afterward servlet lifecycle comes into play.

- The process goes on until its destruction



1. jspInit():
- called once when the JSP page is first loaded into memory, before processing any client requests.
- It's typically used to perform any initialization tasks required by the JSP, such as setting up resources that are used by the JSP during its execution
- Automatic Invocation: JspInit() is invoked by the container, developers don't directly call its method

2. _jspService():
- method that handles client requests and generates responses. It's called every time a request is made to the JSP page.
- The JSP page's main content, including HTML, JSP tags and Java code embedded within <% %> scriptlets, is translated into the _jspService() method.
- _jspService() method is implicitly created & called by the JSP container, developers should not override it

3. jspDestroy():
- called once before JSP is unloaded from memory, typically when the app.

vision

is shut down or when the JSP is modified & needs to be recompiled
- It's used to release resources that were allocated in jspInit(), such as closing database connections or cleaning up other resources.
- JSP container calls jspDestroy() automatically, developers donot invoke it directly.

* List out some advantages of using JSP.

→ It doesnot require advanced knowledge of java
- It is capable of handling exceptions
- Easy to use & learn
- It can tags which are easy to use & understand
- Implicit objects are there which reduce the length of code.
- It is suitable for both java & non-java programmer

vision

* What is the difference b/w include directive & include action?

Aspects	include Directive	include Action
Syntax	<%@ include file = "filename.jsp" %>	<jsp:include page = "filename.jsp" />
When content is included	During JSP compilation time (statically)	At request time (dynamically)
Effect on changes in included file	Changes to the included file require re-compilation of the main JSP to reflect updates	Changes are immediately reflected without re-compilation
Scope of variables	shares the same scope & context as the main JSP	Variables defined in the included file are not accessible in the main JSP unless passed explicitly
Use Case	Best for including static content that doesn't change often, like headers or footers	Suitable for including dynamic content that may change b/w requests, such as data driven components
File extention Support	only includes .jsp files	Can include other file types

* List two methods of JSP life cycle which can be overridden.

→ Two overridden methods are:

1. jspInit():
- called once when JSP page is initialized, similar to the init() in servlet
- overridden to perform any setup tasks or resource allocation needed when the JSP page first loads.

2. jspDestroy():
- called once when the JSP is about to be removed from memory, similar to the destroy() method in servlet
- typically overridden to release resources or perform cleanup tasks.

* Write about JSP Scripting Elements, explain its subdivisions of scripting elements in JSP.

→ JSP scripting elements are one of the most vital elements of a JSP code.
- These <% %> tags contain JSP scripting elements

- only this code will convert to java code. Code other than this is plain or HTML text.

- The scripting elements this help to embed java code to the HTML; CSS, JavaScript code.

→ Subdivisions of scripting elements in JSP:-

1. Scriptlet Tags:-

- <% ____ %>
- JSP scriptlets help in embedding java in HTML for JSP code.
- The code inside a scriptlet tag goes to the _jspService() method of the generated servlet for processing the request.
- for each request, _jspService() method will invoke.

2. JSP Expressions:-

- Expression Tag is used to print out java language expression that is put b/w the tags.
- An expression tag can hold any java language

expression that can be used as an argument to the out.print() method.

→ Syntax: <%= "Java Expression Tag" %>

Ex:- <% "K=" + k %>

3. JSP Declaration Tag :-

- W.K.T. at the end a JSP is translated into Servlet class, so when we declare a variable / method in JSP inside Declaration Tag, it means the declaration is made inside the servlet class but outside the service method.

- Static member, instance variable & methods can be declared inside Declaration Tag.

- Syntax: <%! declaration %>

- Ex:- <%! int count=0; %>

4. JSP comment tag :-

- Comment increase the understanding of the code.
- These comments are only seen in JSP pages & not included in the servlet source code during the translation phase.

→ Syntax: <%-- Comment lines --%>

→ Ex:- <%-- Comment is not visible to the page source --%>

5. Directive Tag :-

- The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet

- <%@ directive attribute = "value" %>

→ 3 types of JSP directives

1) Page Directive
2) Include Directive
3) taglib Directive

* What is JSP Declaration tag ? Explain it with an example.

→ JSP Declaration tag allows you to declare variable & methods in JSP page. Unlike scriptlets, which place code within the page's service method, code within a declaration tag is placed outside the service method.

- Syntax: <%! declaration %>

→ Characteristics :-

- Variables or methods declared in the declaration tag are initialized once & remain accessible across all requests in the JSP page.

- Because they are defined at the class level, declared elements are retained in the JSP's generated servlet class & aren't reinitialized with each request.

- It is often used for defining utility methods, counters, database conn. obj that need to persist throughout the JSP's lifecycle.

→ Ex:- Declaring a variable

```
<%! int counter = 0; %>
<html>
<head>
<title> Page View Counter </title>
</head>
<body>
    <% counter ++; %>
    <h2> This page has been viewed
        <%= counter %> times. </h2>
</body>
</html>
```

→ Counter variable is declared with <%! ... %> & is shared across all requests, so each time the page reloads, the counter increments by one.

* Difference between JSP Scriptlet & Declaration tag.

Aspect	Scriptlet Tag	Declaration Tag
Purpose	Contains code executed each time the page is requested.	Declares variables or methods that persist across requests
Placement In servlet	code is placed inside the service method of the generated servlet	code is placed outside the service method at the class level

Scope	limited to the request	Persistent for the JSP lifecycle
Syntax	<% Java Code %>	<%! Java Code %>
Usage	Used to execute code on each request	Used to declare class-level variables or helper methods
• Ex:-	<% int count = 0; %> (resets on each req.)	<%! int count = 0; %> (retains across all requests)

* Classification of JSP Directives:-

→ Classification:

1. Page Directive
2. Include Directive
3. taglib Directive

1) Page Directive

- The page Directive defines attributes that apply to an entire JSP page.
- Syntax:- <%@ page attribute = "value" %>
- Attributes of JSP page directives:

a) import
b) language
c) Page Encoding
etc

Ex: a) import attribute

- defines that the specified set of classes or packages must be imported.

Ex <%@ page import = "java.util.Calender" %>

b) language attribute

- defines which scripting language is to be used in JSP page

Ex: <%@ page language = "java" %>

2) Include Directive

- It works similarly to the # include of the C preprocessor directory.
- Its work is to merge the content of another file into .jsp source input stream at the translation phase

- syntax:- <%@ include file = "filename" %>

3) Taglib Directive

- The taglib directive specify libraries of tag files we will use in the current JSP file. This can either be a Custom JSP tag library or JSP Standard Tag Library (JSTL).

- The beauty of the tag file is that you can invoke them with attributes, from within JSP pages & get some results back.

- You may code many taglib directives anywhere in your JSP page, but the taglib directives are normally coded at the top of the JSP page.

→ Syntax: <%@ taglib attributes %>

- Declares a custom tag library or a Standard Tag library to use in this page