CS -29 MAJOR - 11
Advance Java Programming(J2EE)

Presented by : Dhruvita Savaliya

UNIT 1 - Introduction to J2EE and JDBC

Topics of Introduction to J2EE

- Introduction to J2EE
- Enterprise Architecture Styles:
  - Two-Tier Architecture
  - Three-Tier Architecture
  - N-Tier Architecture
- Enterprise Architecture
- The J2EE Platform
- Introduction to J2EE APIs (Servlet, JSP, EJB, JMS, JavaMail, JSF, JNDI)
- Introduction to Containers
- Tomcat as a Web Container

Page number 5 to 50

Topics of Introduction to JDBC

- JDBC Architecture
- Types of JDBC Drivers
- Introduction to major JDBC Classes and Interface
- Creating simple JDBC Application
- Types of Statement (Statement Interface, PreparedStatement, CallableStatement)
- Creating CRUD Application

Page number 51 to end

Basics of JAVA :

- Java was developed by James Gosling, an engineering of Microsystems in 1991.
- Java was previously known as “Oak” but it was an existing language so there was a need to rename it.
- It was renamed with “JAVA” in 1995. Java was the Latin name of the coffee beans.
- It was use to developed consumer based application such as Microwave, remote control, TV etc…

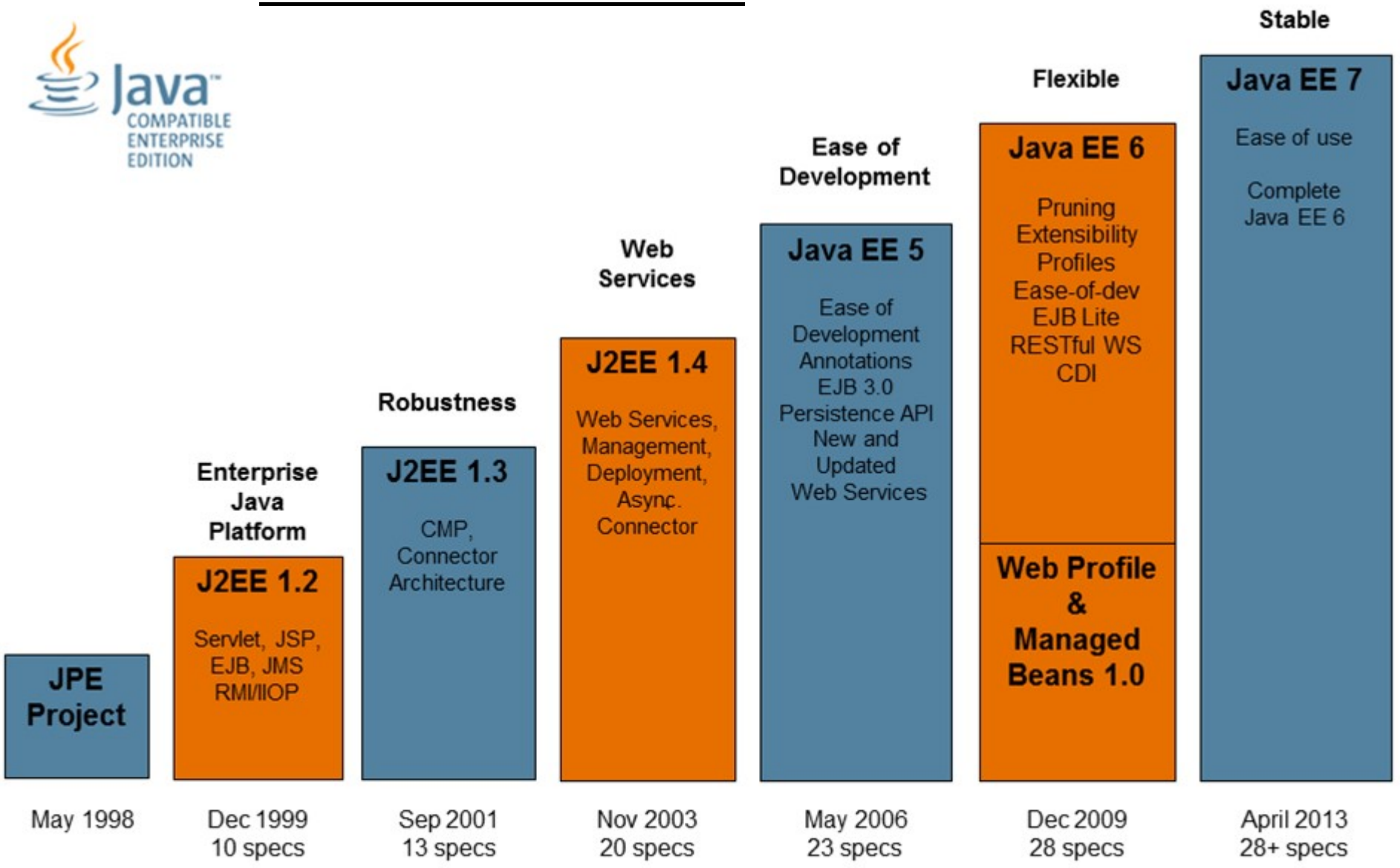
Introduction to J2EE

- J2EE stands for **Java 2 platform Enterprise Edition**.
- It is used to develop multi tier architecture based applications.
- It is used to reduces the cost and complexity of developing multi-tier services.
- It can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures.
- J2EE is an open standard which is provided by **Sun Microsystems** for application which run on servers.
- It provides multi-tired architecture for commercial applications.
- J2EE is used in order to maintain Speed, Security and Reliability of Server-Side technology.

- It includes J2SE+ most of the other java technologies including JavaMail, Servlets, JSF, JMS, EJB and others. Most of the API is component- oriented.
- They are used to provide interfaces for business oriented components for enterprise and distributed internet applications.
- Java has emerged as one of the most of the mature and commonly used programming language for building enterprise software.

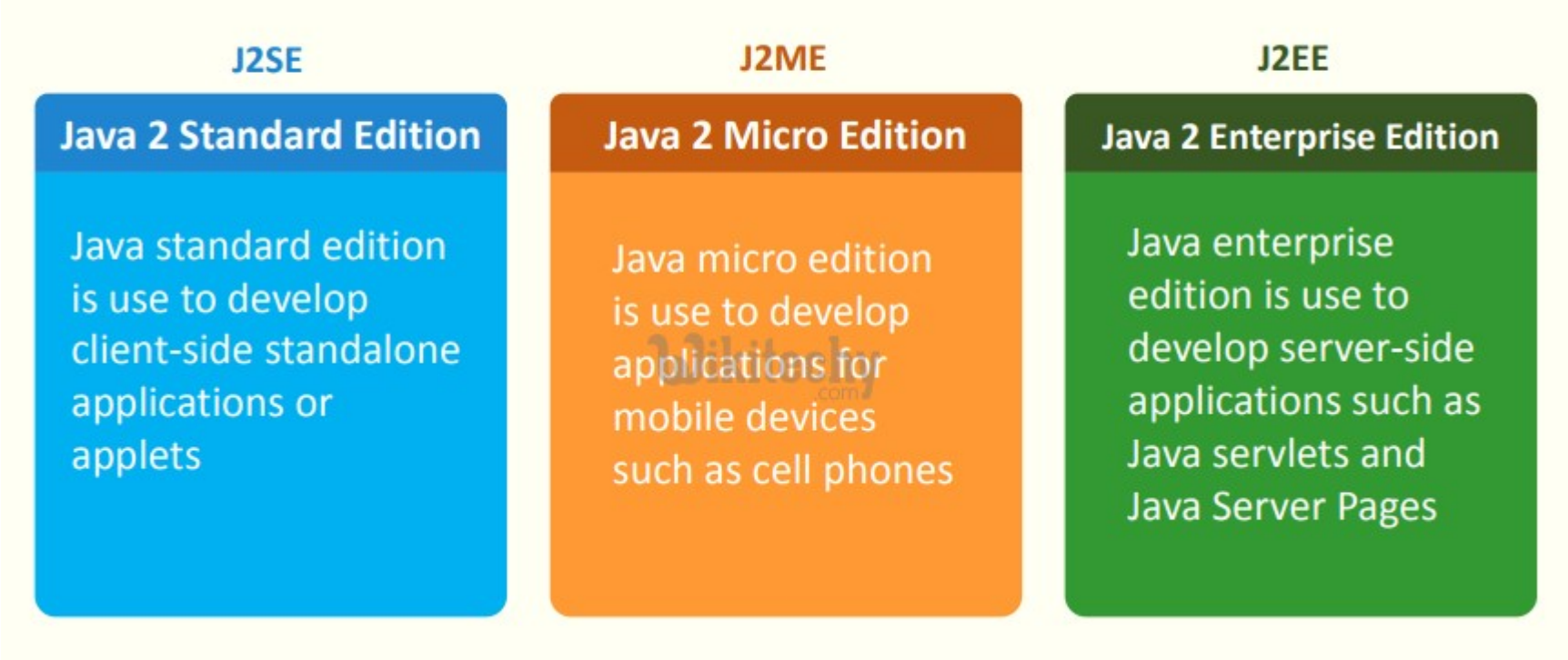
- Some characteristics of enterprise applications are :
- Enterprise applications can be run on different platforms supporting the java 2 platform.
- Enterprise applications are portable between application servers supporting the J2EE specification.
- So, J2EE is Open and standard based platform for developing, deploying and managing n-tier, Web-enabled, server-centric, and component-based enterprise applications.
- Enterprises today need to extend their reach, reduce their costs, and lower their response times by providing easy-to-access services.

Versions of J2EE



Date	Correct	Incorrect	Incorrect
1999	J2EE 1.2	Java EE 1.2	Jakarta EE 1.2
2001	J2EE 1.3	Java EE 1.3	Jakarta EE 1.3
2003	J2EE 1.4	Java EE 1.4	Jakarta EE 1.4
2006	Java EE 5	J2EE 1.5	Jakarta EE 5
2009	Java EE 6	J2EE 1.6	Jakarta EE 6
2013	Java EE 7	J2EE 1.7	Jakarta EE 7
2017	Java EE 8	J2EE 1.8	Jakarta EE 8
2019	Jakarta EE 8	J2EE 1.8	Java EE 8
2020	Jakarta EE 9	J2EE 1.9	Java EE 9

Platform/Editions of JAVA :



Introduction to Enterprise application Design Framework :

- Enterprise Application Design is divided into **6 logical layers**, which is related to logic of the client tier, middle tier, and database tier. It defines which layer belongs to which tier.
- **(1) Presentation Manager :**

  The presentation manager defines the user interface. It is always reside on client tire. It manage the information display to the user.
- **(2) Presentation Logic :**

  The presentation logic defines the navigation system of the user interface, how and what will be displayed to the user.

  It may reside with the client tier or business tier or database tier based on thin client and thick client and application tier.

- **(3) Application Logic :**

  Application logic defines actual application logic with it. Application logic can be connectivity validations etc..

  It may reside with the client tier or business tier or database tier based on thin client and thick client and application client.
- **(4) Business Logic :**

  The business logic layer contains the business rules of the application.

  It should be shared with the whole application. It may reside with the business tier.
- **(5) Database Logic :**

  The database logic defines the table structure and the relationship between the tables.

  It also includes all the constraints of the table.

  It always reside on the database tier.
- **(6) Database Manager :**

  The database Manager stored the persistent data.

  It always reside with database tier.

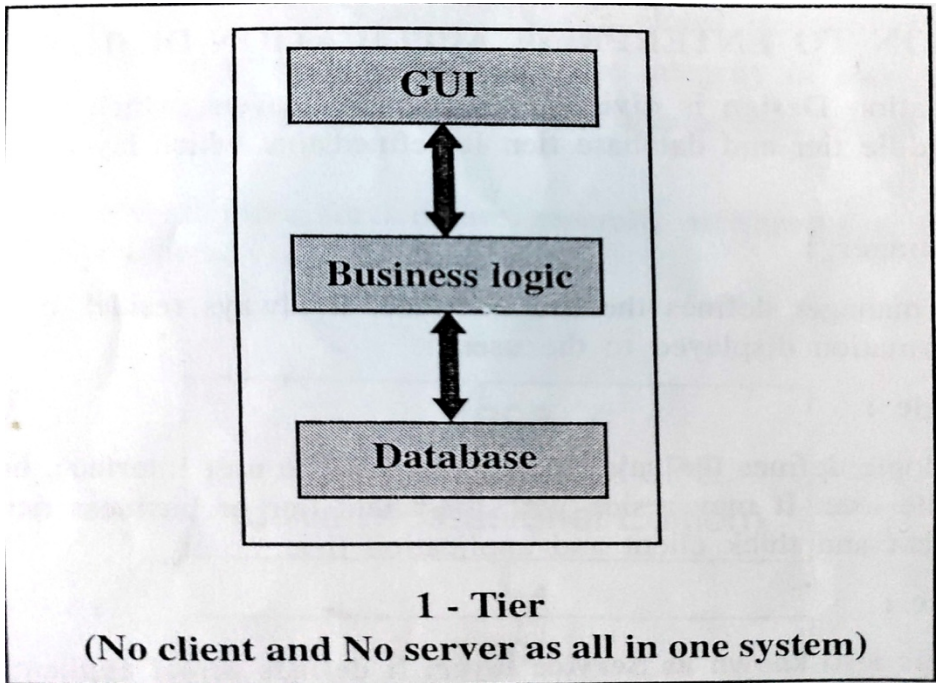
Enterprise Architecture Styles:

- There are main Four Architecture Style in J2EE :

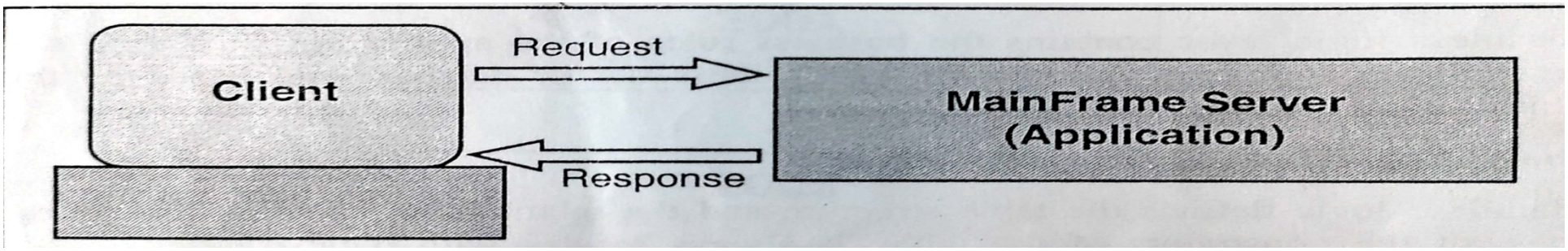
1. Single Tire Architecture
2. Two Tire Architecture
3. Three Tire Architecture
4. N-Tire Architecture

Single Tire Architecture :

- Simple software applications are written to run on a single computer.
- All user input, verification, business logic and data accessed could be found together.
- This kind of architecture is called as Single tier Architecture, because all logic application services, the presentation, business rules and the data access layers exist in a single computing layer.



In client/server architecture there is a dumb terminal (client) only works for I/O and all the layers reside at Server (mainframe).



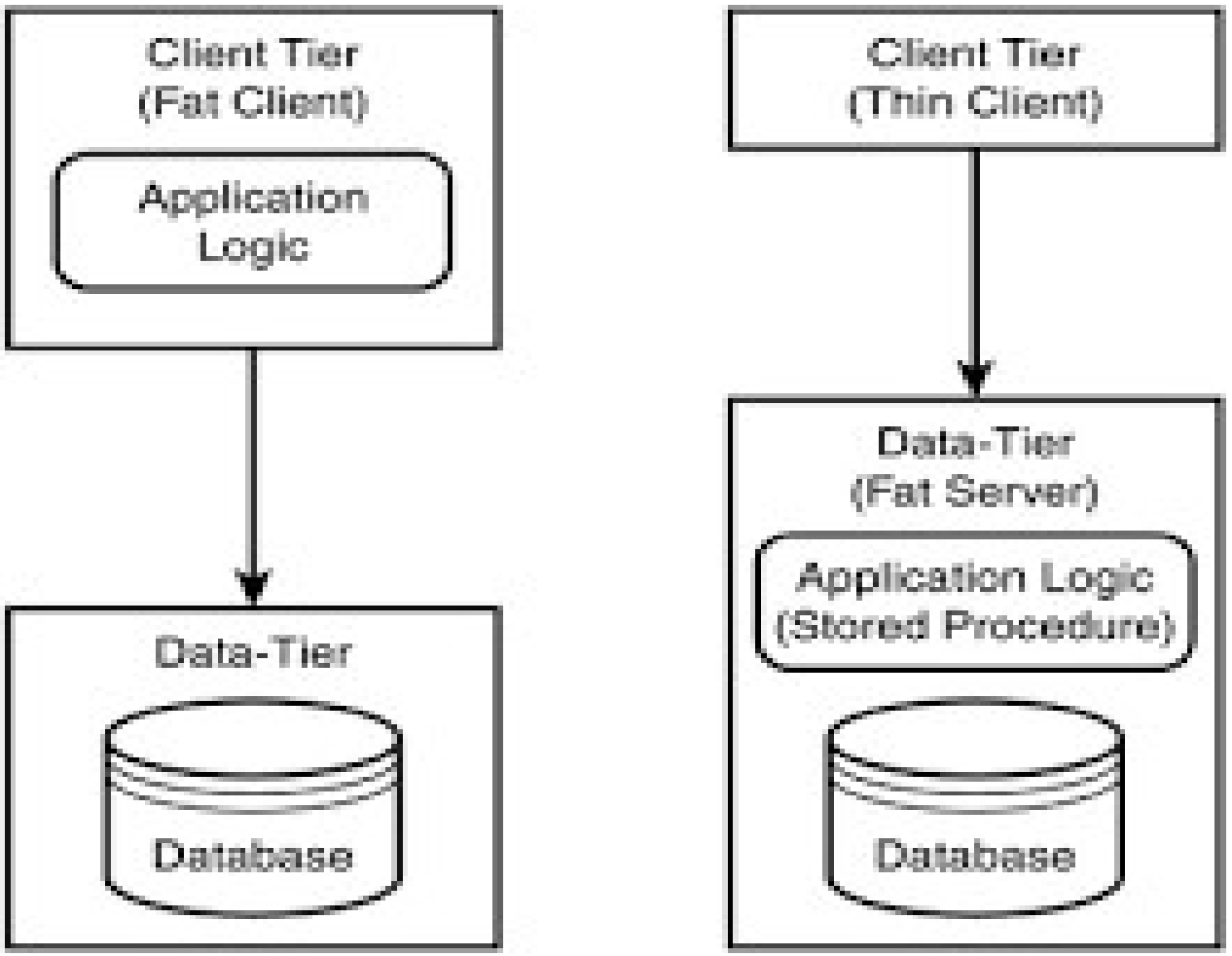
Advantage: Single tier system is relatively easy to manage & data consistency is simple because data is stored in only one single location.

Disadvantage : However in the world now a day single storage location is not sufficient because of the changing the business needs.

With the single tier application we cannot share the data large in the large amount, it can also not handle multiple users. Because of these many reasons two tier Architecture required.

Two Tier Architecture :

- Application which is divides into two separate tiers client machine and database server machine this kind of application is called as two tier architecture application.
- The application includes the presentation and business logic.
- Data is accessed by connecting client machine to a database which is lying on another machine.

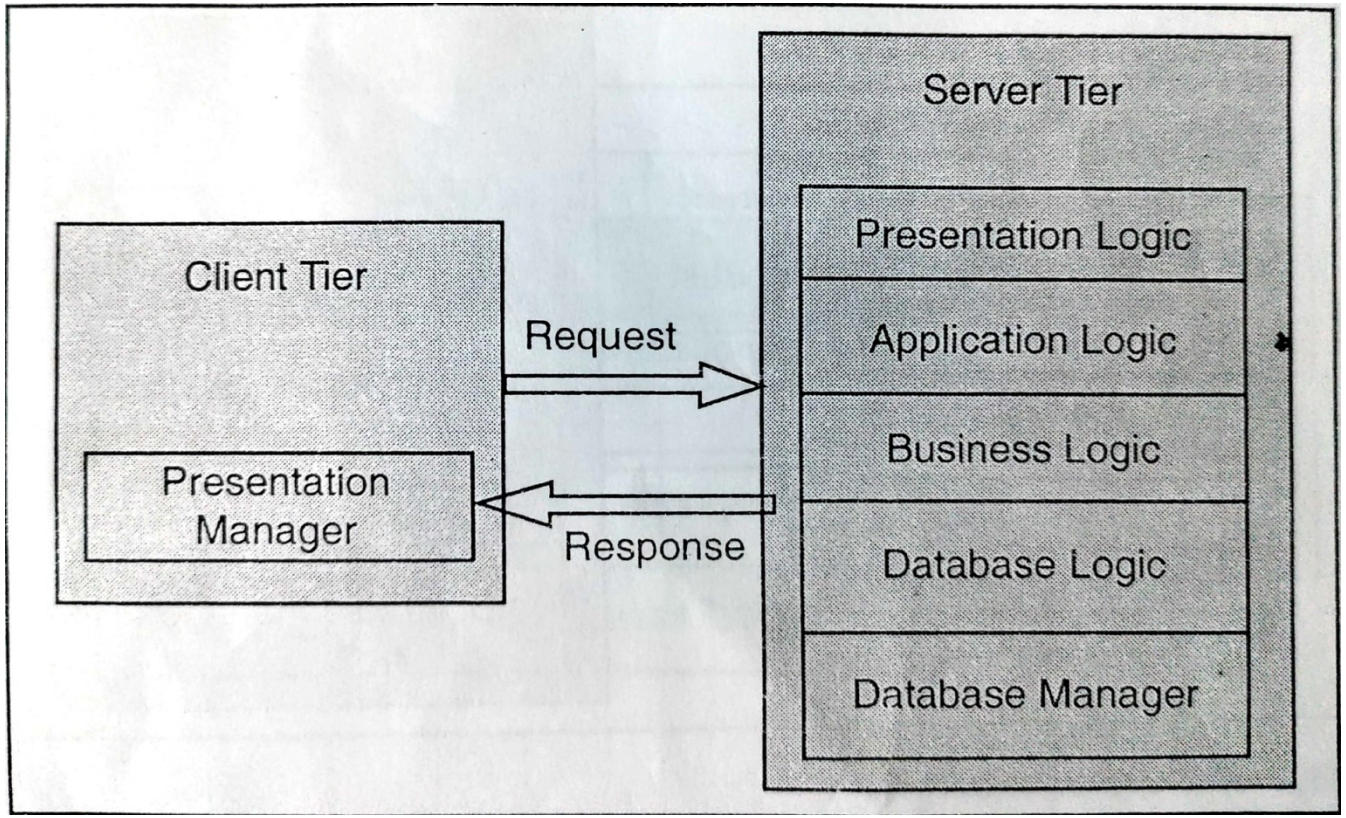


2-Tier Architecture
(Client/Server)

- **Thin Tier :**

  With the two tier architecture, if the presentation manager reside only with the client tier then the client is called as Thin Client.

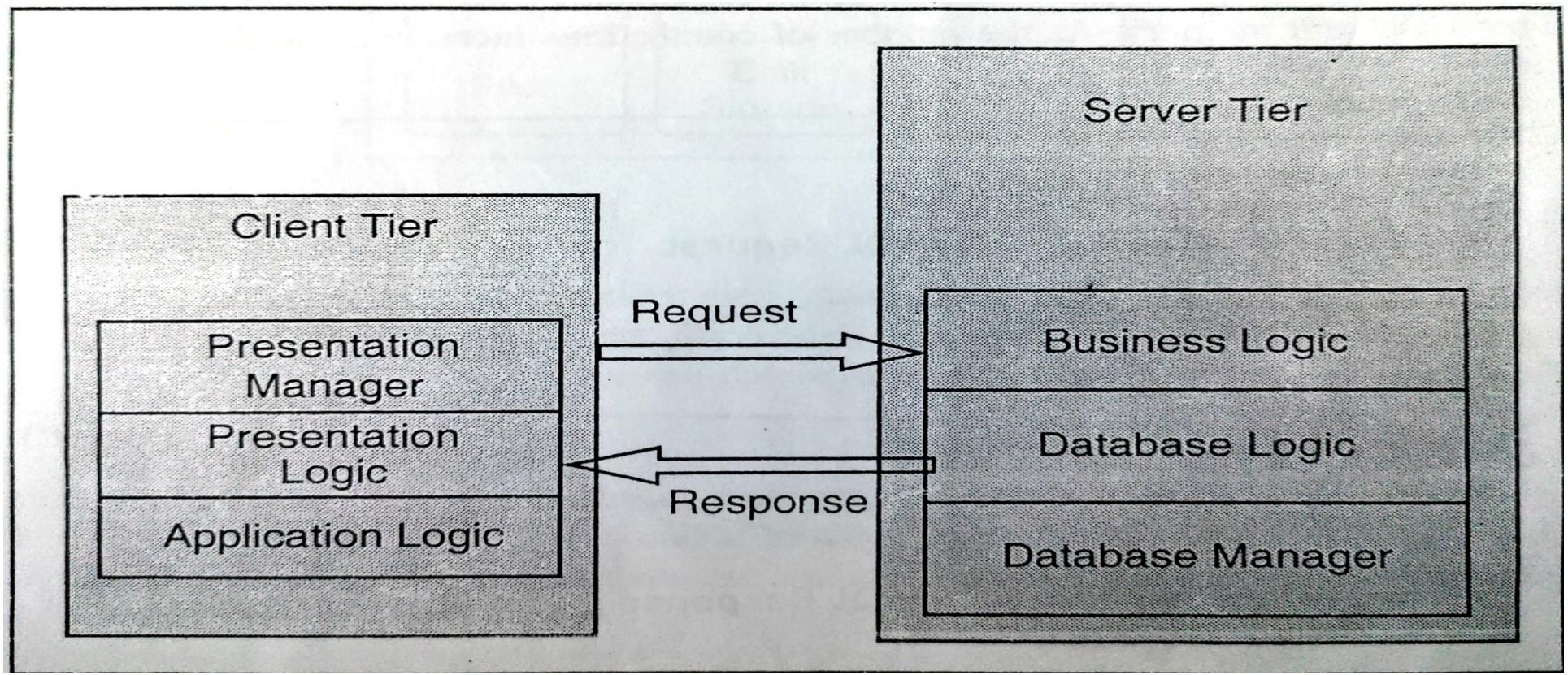
  Other presentation logic , application logic, business , database logic and database manager reside with the server side.



- **Thick Tier :**

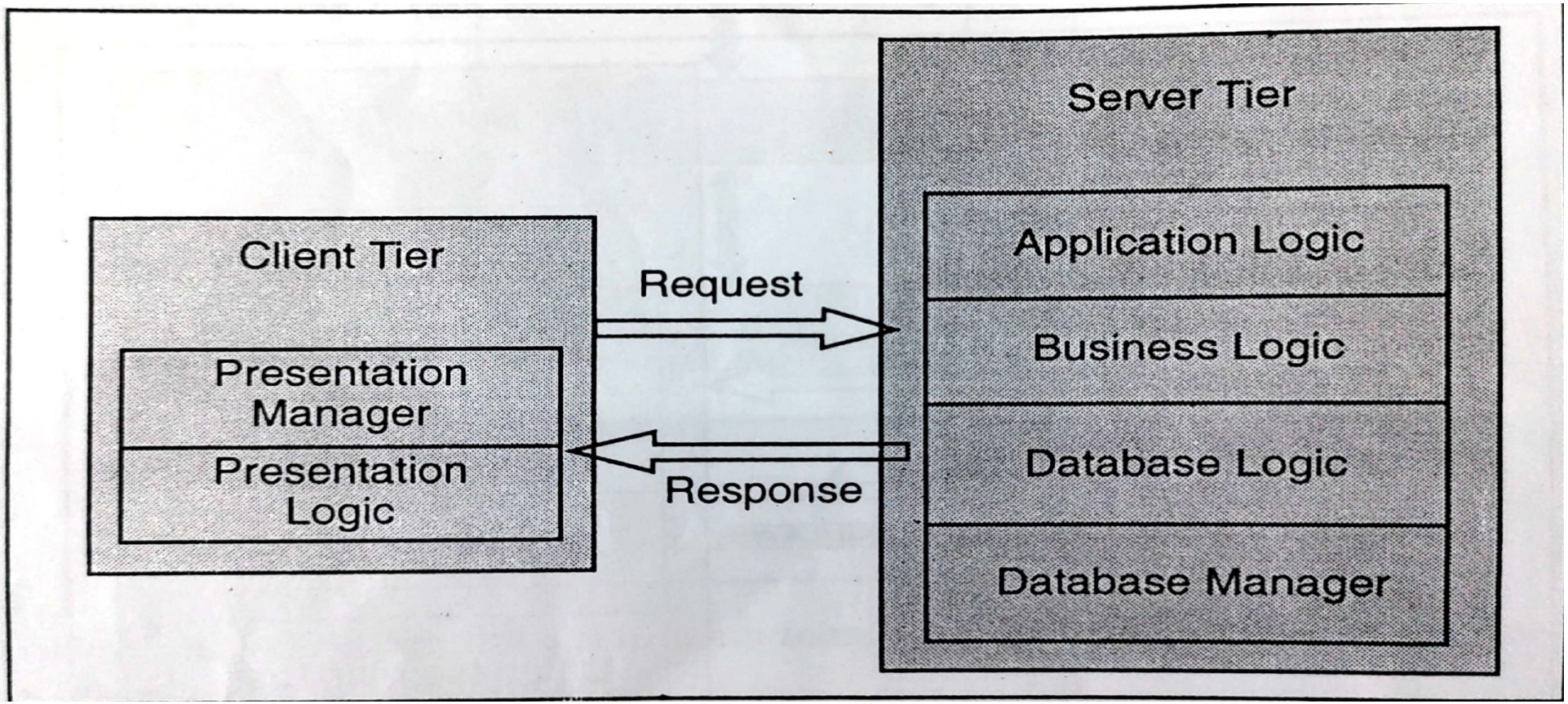
  With the two tier architecture if the presentation manager, presentation logic , application logic reside with the client tire is called Thick Client.

  Others like business logic, data logic and database manager reside with the server side.



- **Normal Client :**

  With the two tier architecture if the presentation logic reside with the client tier then the client is called Normal Client. Other like application logic, business logic, data logic and database manager reside with the server side.



- **Advantage :**

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

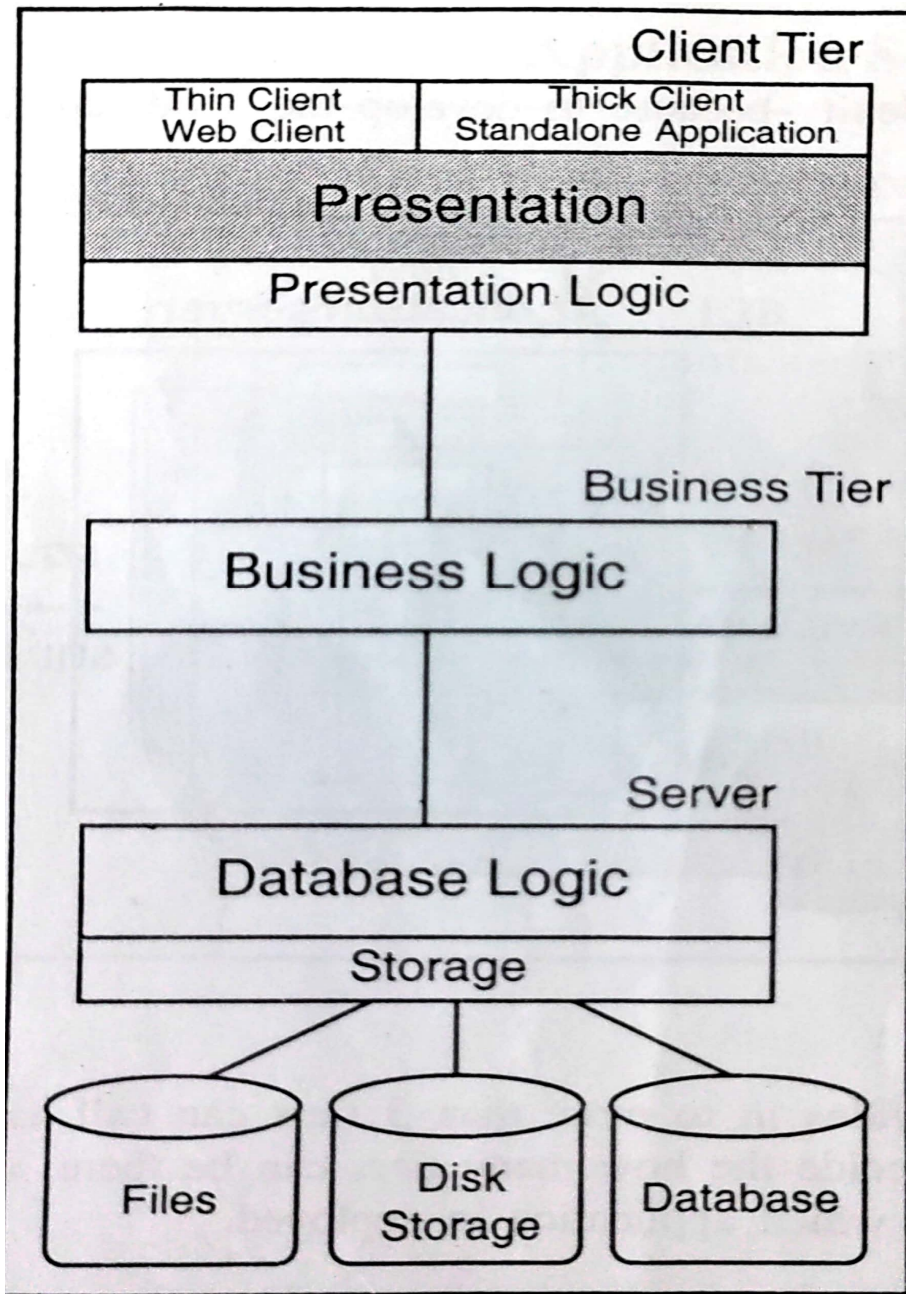
  One of the disadvantages of two Tier Architecture is that the application is expected to support a limited number of users.

Three Tier Architecture :

Application which is divided into three tires Client Tire, Middle Tire, Database Tire(Enterprise information system) is known as Three tire architecture application.

Logic physically separates the business rules.

The presentation layer and logic runs on client machine, application and business logic runs on J2EE server and database logic is there with database layer.


Client Tier
Thin Client
Web Client
Thick Client
Standalone Application
Presentation
Presentation Logic
Business Tier
Business Logic
Server
Database Logic
Storage
Files
Disk
Storage
Database

- **Thin Client :**

  With the three tire architecture if the presentation manager resides only with the client tire then the client is called as Thin Client.

  Presentation logic, application logic and business logic are with the business tire and database logic and database manager are with the EIS tire(database).

- **Thick Client :**

  With the three tire architecture if presentation manager, presentation logic, application logic reside with the client tire then the client is called as Thick Client.

  Business logic is only with the business tire . The database logic and database manager are with the EIS tire (database).

- **Advantages of Three Tier Architecture :**

  It improves scalability since the application server can be deployed on many machine.

  The database no longer requires a connection from every client it only requires connections from a smaller number of application server.

  It provides security because client does not have direct access to the database.

- **Disadvantages of Three Tier Architecture :**

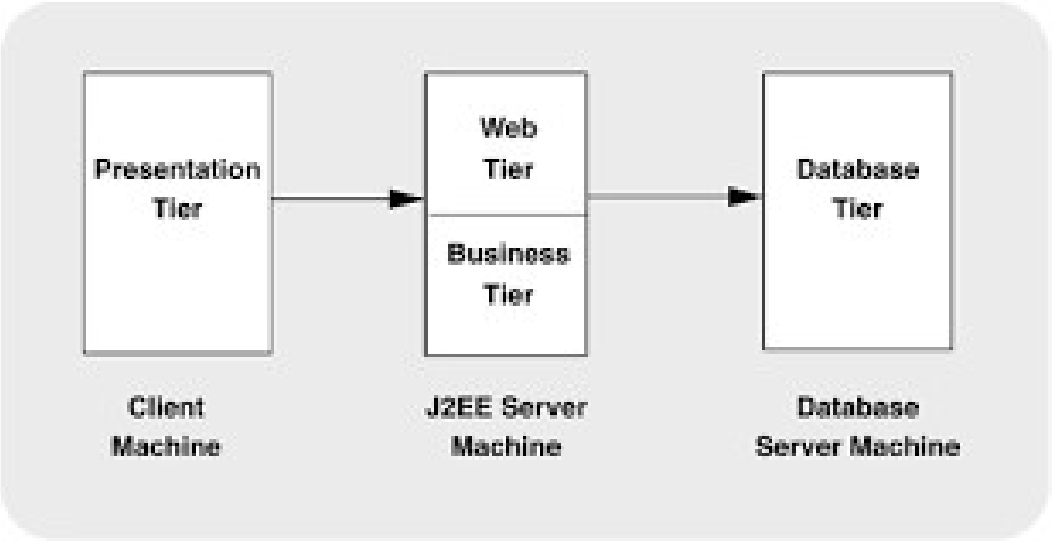
  It increases the complexity because to develop the three tire application is more difficult than developing a two tire application.

N Tier Architecture :

An application which is divided in to more than three tire can be called N-Tire architecture.

In N Tire architecture it is not decided how many tires can be there, it depends on computing and network hardware on which application is deployed.

Basically it is divided into four application layers.



1. Client Tier :

Client tire consists of the user interface for user request and print the response.

It basically uses the browser or applet as client side applications.

2. Web Tier :

Web tire consists of the JSP and servlets dynamic web pages to handle the HTTP specific request logons, session, access the business service.

Finally server is constructs a response and send it back to the client.

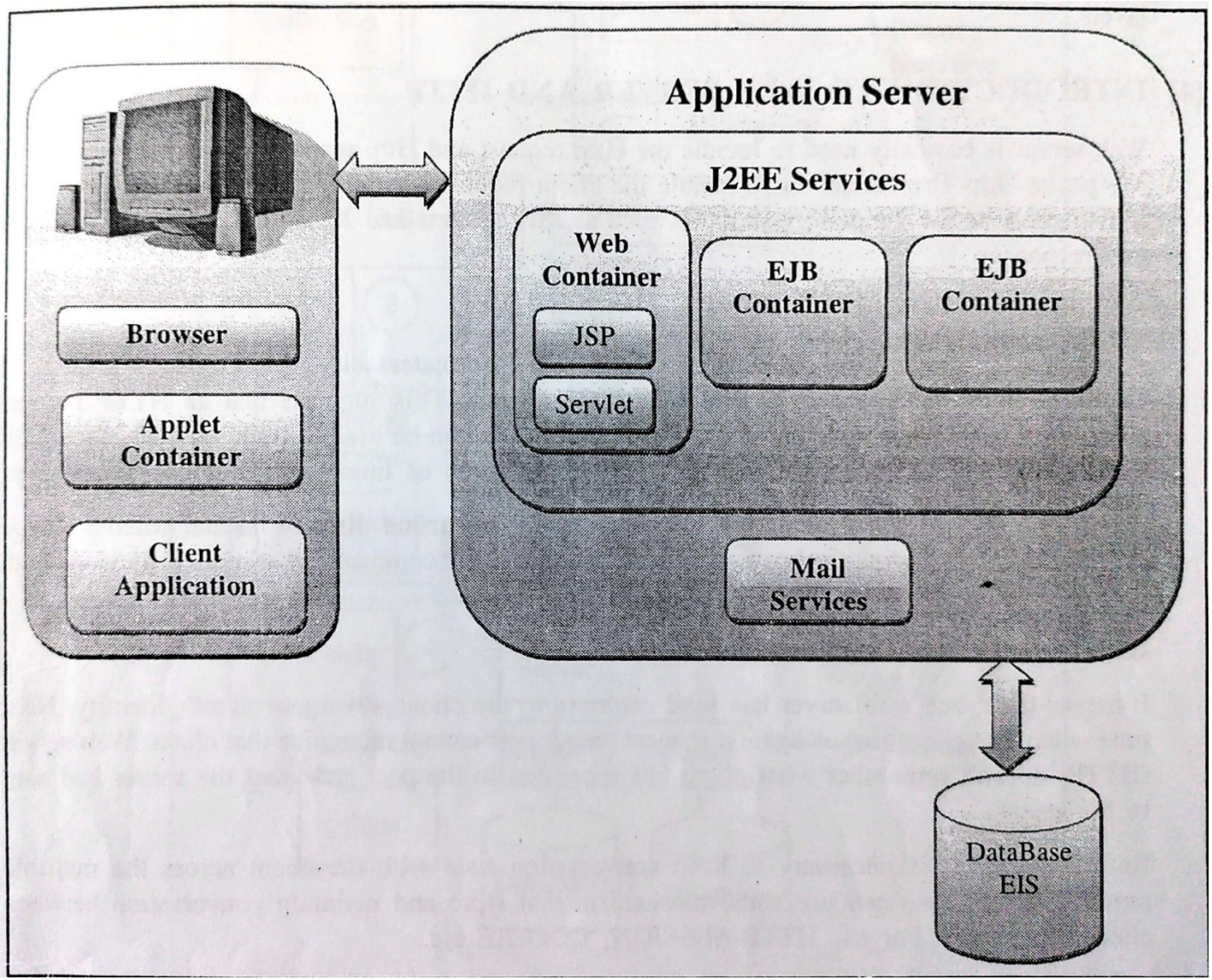
3. Business Tier :

Business tire can consists of the business logic for the J2EE application.

For Example : EJB (Enterprise Java Bean)

The benefit of having a centralized business tire is that same business logic support different type of clients like browser, WAP, other stand-alone applications etc..

- **EIS Tier (**Enterprise Information Systems**) or Database Tier :**
  EIS tier consists of the DBMS/ RDBMS.
  It handles the users SQL Request and generates appropriate response based on queries.
  It is responsible for communicating with external resources such as legacy systems, ERP systems; messaging systems like MQSeries etc.
  It also stored all persistent data in the database.
  J2EE is based on n tier or multi tier architecture applications.
  J2EE makes easy to develop the Enterprise Application based on 2, 3 or more application layers.
  Here it also proves that the J2EE is distributing computing framework and multi tiered application.
- **Advantages of Three Tier Application :**
  Separation of user interface logic and business logic is done.
  Business logic resides on small number of centralized machines (may be just one).
  Easy to maintain, to manage, to scale, loosely coupled etc.
  Additional features can be easily added.
- **Disadvantages of Three Tier Application :**
  It is having more complex structure and difficult to setup and maintain the all the separated layers.


Application Server
J2EE Services
Web Container
JSP
Servlet
EJB Container
EJB Container
Mail Services
Browser
Applet Container
Client Application
DataBase EIS

- **Web Server and HTTP :**

  Web server is basically used to handle the Http request and Http response.

  In J2EE web server is Apache Http Tomcat server to handle the client request and send the response to the client.

  It also contain the dynamic web pages such as JSP servlets and XML to handle such requests and responses.

- **HTTP :**

HTTP is the protocol that allows web servers and browsers to exchange data over the web.

It is a request and response protocol.

The client requests a file and the server responds to the request.

HTTP uses reliable TCP connections—by default on TCP port 80

In HTTP, it's always the client who initiates a transaction by establishing a connection and sending an HTTP request.

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

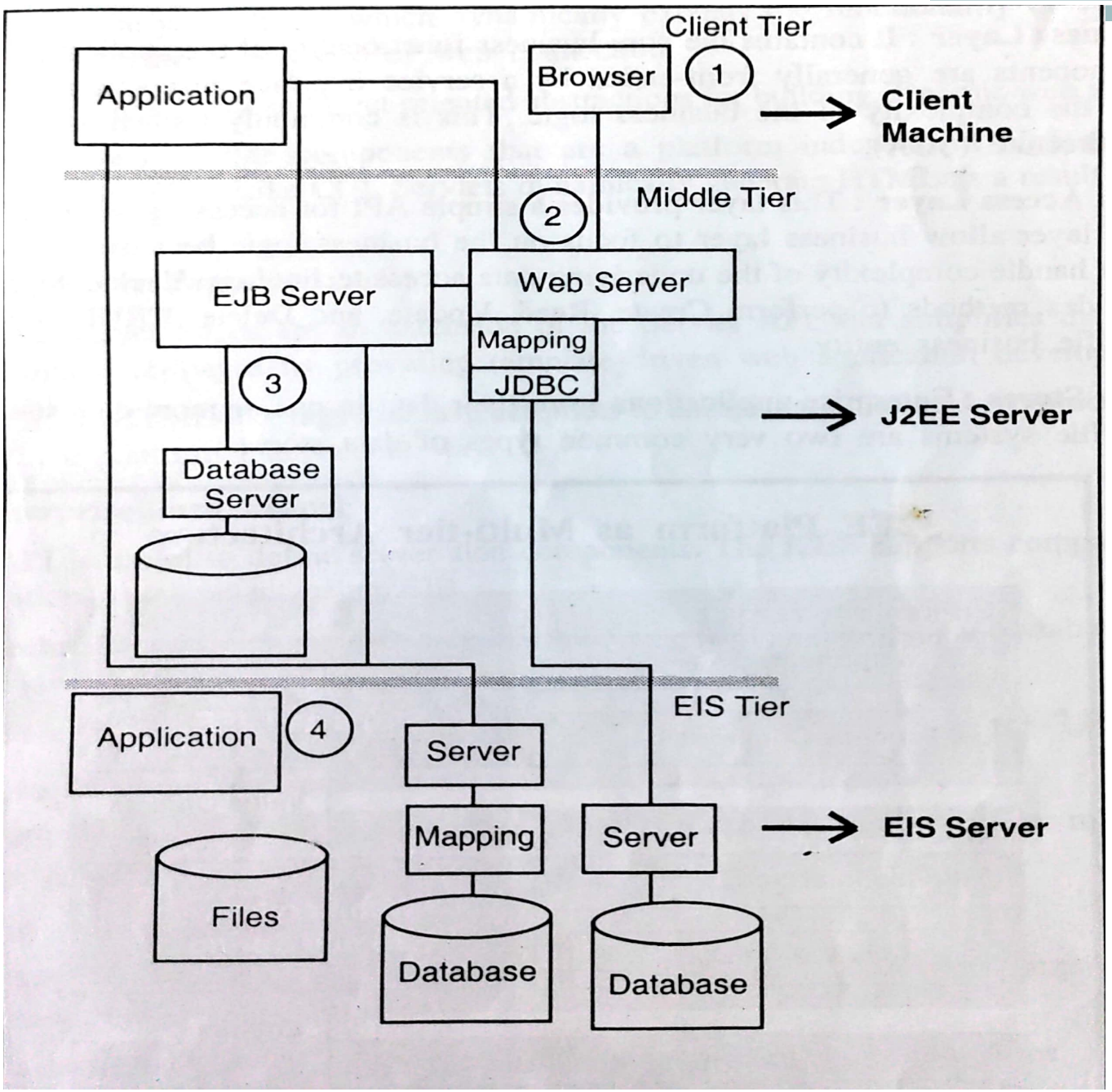
When we use a web browser, we do not enter HTTP command directly instead when we type URL or click a hyperlink, the browser action into the HTTP commands

- **HTTP as a Stateless Protocol :**
  protocol is known as stateless protocol because when client requests the web server and once server has sent to response to the client.
  But sometimes it is necessary to keep conversation state with the client across the multiple requests. For that we can use some mechanism that store and maintain conversation between client and server. For ex, HTTP SESSION, COOKIE etc..

Enterprise Architecture of J2EE :

- Java to enterprise edition is basically developed for Commercial Projects and web solutions Business Solutions for commercial project is solved using multitier architecture.
- The J2EE platform uses a multi-tiered distributed application model for enterprise applications.
- By dividing Application logic into the various components according to its task or function and the various application components that are gathered as a J2EE application are installed on different machines depending on the tier in the multi-tiered J2EE environment to which the application component belongs.
- Following tiers are available in J2EE:

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



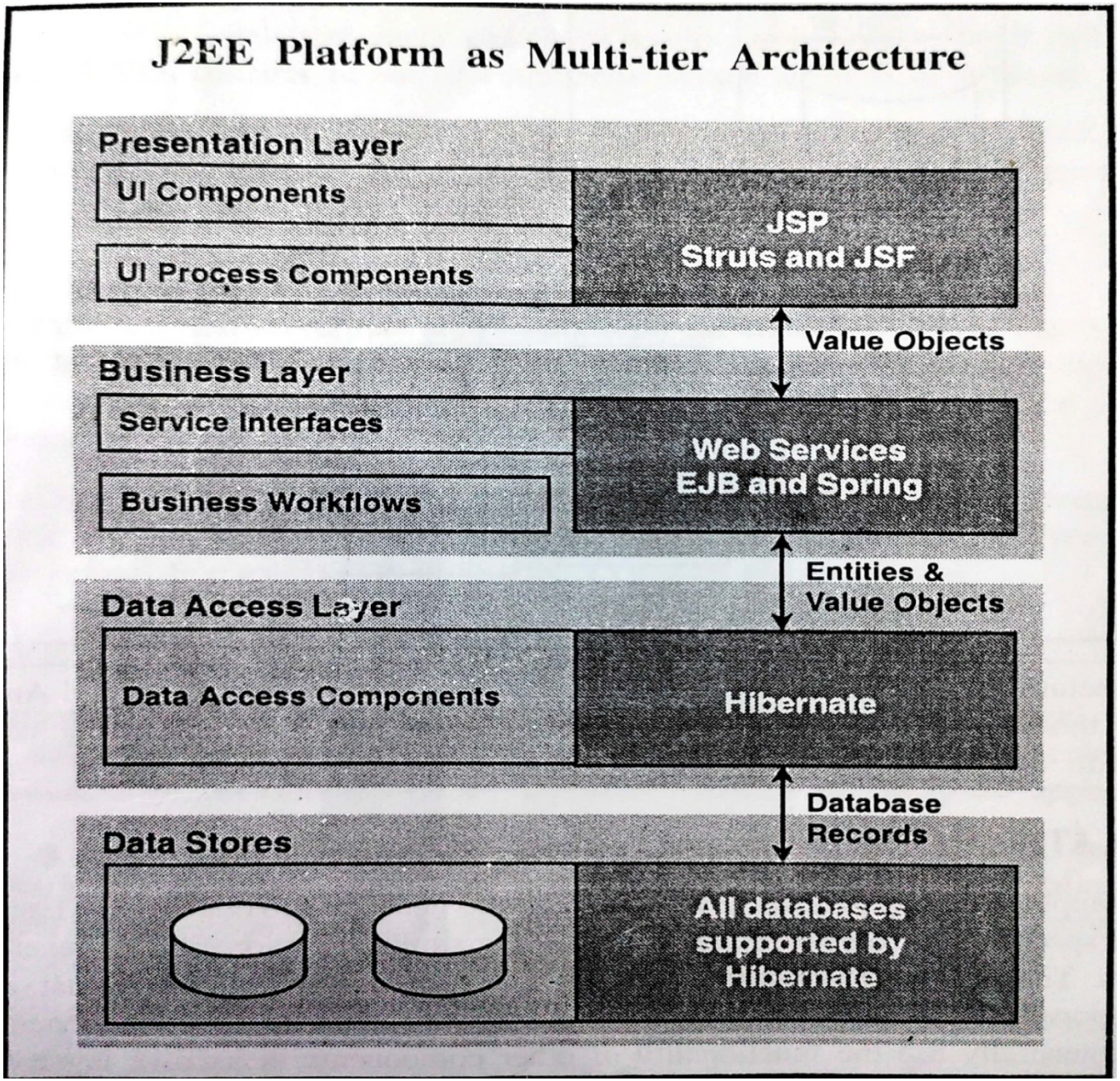
J2EE Architecture is dividing into the three or four tier it is known as Multi-tiered Architecture.

But actually those four tires are divided into the three locations:

(1) Client Machine
(2)J2EE Server
(3)EIS Server.

J2EE Platform :

- Enterprise applications are built using some components connected to one another.
- Each component provides a specific functionality.
- Components having similar functionality are generally grouped into layers.
- These layers are further arranged as a stack in hierarchical format, where lower layer components provide services to the higher layer components.
- A component in a given layer will generally use the functionality of other components in its own layer or the layers below it.


J2EE Platform as Multi-tier Architecture
Presentation Layer
UI Components
UI Process Components
JSP
Struts and JSF
Value Objects
Business Layer
Service Interfaces
Business Workflows
Web Services
EJB and Spring
Entities &
Value Objects
Data Access Layer
Data Access Components
Hibernate
Database
Records
Data Stores
All databases
supported by
Hibernate

Introduction to J2EE API's :

- The J2EE platform provides a set of APIs to develop the different type of applications.
- It uses the concept of distributed computing to Develop the enterprise application.
- The API is refered as "Application Programming Interface" .

1. Servlet
2. JSP (Java Server Page)
3. EJB (Enterprise JavaBeans)
4. JMS (Java Messeging Service)
5. JavaMail
6. JSF (Java Server Faces)
7. JNDI (Java Naming and directory Interface)
8. RMI (Remote Method Invocation)

1. Servlet:

Servlets are the java classes which dynamically extends the functionality of Web Server. It executes on the server side of web connection.

The Java Servlets provides object oriented abstractions for building dynamic web applications.

Servlets are server side components that are a platform independent high performance replacement for CGL Like CGI, Servlets dynamically generate HTML as a result of a HTTP request.

2. JSP [Java Server Pages]:

The Java Server Pages are an extension of the Servlet API that simplifies the generation of dynamic web pages by providing template driven web application development.

It uses XML/HTML-like tags and Java scriptlet to encapsulate the logic required to generate pages in a platform independent way.

3. EJB [Enterprise Java Beans]:

This API is useful to define server side components.

The J2EE supports components based application development using EJB.

This technology enables rapid & simplified development of distributed, portable application based java technology. There are 3 main types of EJB's:

1. session beans
2. entity beans
3. message driven beans.

- **4. Java Mail:**

  This API provides a platform independent and protocol independent framework to build Java based email applications.

  It uses MIME, POP3 and SMTP as its underlying transport mechanisms.

- **5. JSF [Java Server Faces]:**

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

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

- **6. JMS [Java Message Service]:**

  The JMS API is messaging standard that allow J2EE application components to create, send. receive & read Msg. in distributed Environment.

- **7. JNDI [Java Naming & Directory Interface):**

  JNDI standardizes access to different types of naming services.

  The API is designed to be independent of any specific naming or directory service implementation.

  Connection pool is accessible by using JNDI.

- **8. JTA (Java Transaction API) :**

  JTA is a specification that enables distributed transactions across multiple resource managers, like databases, ensuring data consistency and integrity. It allows applications to manage transactions that span across different databases or other resources.

- **9. RMI (Remote Method Invocation):**

  RMI is a specification that enables distributed transactions across multiple resource managers, like databases, ensuring data consistency and integrity. It allows applications to manage transactions that span across different databases or other resources.

Introduction to Container :

There are two types of client one is "Thin client "and other is "Thick Client".

Generally Thin Client in multi-tiered architecture is hard to manage.

But using Component based platform independent architecture of J2EE we can easily write J2EE application because reusable components are stored at Business Logic and easily shared within whole application.

For all these components Server provides Services in Form of Containers.

Because of these Containers we need not to write services for each component included in our application.

Containers are Interface between a Component and Client (Lower Level Code) or platform oriented functionality which supports component.

Container provides Communication platform between client and components.

Types of Container:

There are four basic containers in J2EE.

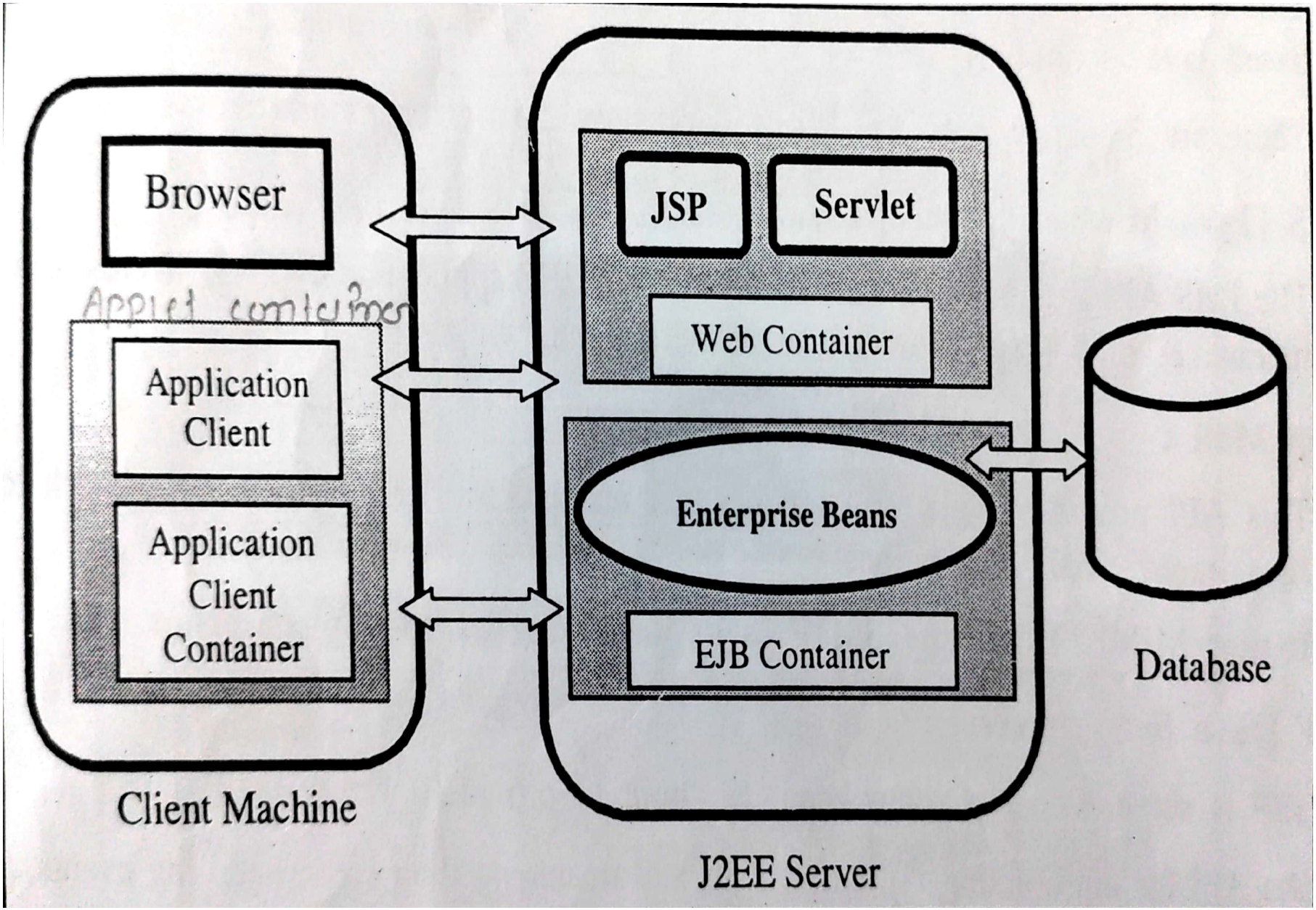
In that two server containers and two client containers are listed below.

1. Applet Container
2. Application Container

Client Containers

3. EJB Container
4. Web Container

Server Containers


Browser
Applet containers
Application Client
Application Client Container
Client Machine
JSP
Servlet
Web Container
Enterprise Beans
EJB Container
J2EE Server
Database

1. Applet Container :

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

Applet container manages the execution of applets, which are Java programs embedded in web pages.

2. Application Container :

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

application container is a runtime environment that provides services and manages the lifecycle of J2EE components, such as EJBs and servlets.

3. EJB Container :

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

EJB is a runtime environment that manages the execution of enterprise beans.

It acts as an intermediary between the business logic within the beans and the application server, providing services like transaction management, security, and resource pooling

4. Web Container :

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

provides the runtime environment for web applications, specifically handling servlets and JSPs. It extends the functionality of a web server by offering services like lifecycle management, security, and concurrency control for web components

Tomcat as a Web Container :

- Apache Tomcat or Tomcat is an open source web server and servlet container developed by the Apache Software Foundation (ASF).
- Tomcat implements the Java Servlet and the Java Server Pages (JSP) specifications from Sun Microsystems.
- Tomcat also provides a "Pure Java" HTTP web server environment for Java code to run in.
- In the simplest config Tomcat runs in a single operating system process.
- The process runs a Java virtual machine (JVM).
- Tomcat provides Multi threading environment.
- Every single HTTP request from a browser to Tomcat is processed in the Tomcat process in a separate thread.
- Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files.
- Tomcat is powerful web container provided by Apache. Using Tomcat we can develop Servlet. JSP application.

- Tomcat is free web container that has can be used to run code of JSP, Servlets.
- Tomcat is also useful to debug JSP and Servlet pages before deploying on the server. Using JSP and Servlets we can develop dynamic web pages.
- Web container is the component of a web server that interacts with Java servlets.
- A web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.
- A Web container handles requests for servlets, Java Server Pages (JSP) files, and other types of files that include server-side code.
- The Web container creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks.

Basic Requirement:

1. Sun Java JDK 5.0 or higher
2. Tomcat 6.0 or higher
3. Eclipse or Netbeans (Editor)

Class Path:

1. To set the Class path follow following steps
2. Right click on My Computer
3. Properties → Advanced → Environment variables → New
4. Variable Name = CLASSPATH
5. Variable Value C:\Program Files\Apache Software Foundation\Apache Tomcat 6.0\lib\servlet-api.jar**;**
6. Then Press OK Button.

Directory Structure:

Directory structure of Tomcat as given below:

Bin: All the Scripts and Batch files are stored into this folder which used to startup Tomcat.

Conf : All the Configurations files for Global and Server Configuration and also contains user Authentication settings.

Log : All the Server Logs are resides in to this folder.

Lib : JAR files are resides in to this folder, which is used by Tomcat.

Webapps: All the JSP and Servlets application's Folders or directories are stored.

Work: All the Temporary and Pre-compiled files are stored in to this folder

[10] SUMMARY

- Enterprise Application Design is divided into the 6 different layers.
  1. Presentation Manager
  2. Presentation Logic
  3. Application Logic
  4. Business Logic
  5. Database Logic
  6. Database Manager

1. An application which runs on the single computer is known as Single Tier Application.
2. The two tier architecture is divided into the Client and Server. Client Application has Business logic and as well as Presentation layer also. Where as Server has Database Services.
3. Three tier architecture is divided into the three section Client Tier, Business Tier/Middle Tier and EIS (Enterprise Information System).
4. N tier architecture is divided into the more than three sections. Basically it is divided in to the four sections: Client Tier, Web Tier, Business Tier and EIS Tier.
5. J2EE Architecture is dividing into the three or four tier it is known as Multi-tiered Architecture. But actually those four tires are divided into the three locations : (1) Client Machine (2) J2EE Server (3) EIS Server.
6. J2EE platform provides different types of APIs for the different types of functionality which are used to develop an J2EE Application. Such like JDBC, RMI, JMS, JNDI, JTA, JavaMail, JSP, EJB etc.
7. Containers are Interface between a Component and Client (Lower Level Code) or platform oriented functionality which supports component. Container provides Communication platform between client and components.

8. Basic Available Containers are Applet Container, Application Container, EJB Container, and Web Container.
9. An application server, in N-tier architecture, is a server that hosts an API to expose business logic and business processes for use by third-party application.
10. The Web Container is a J2EE Server that is used host web applications.
11. Enterprise Java Beans (EJB) components are components of J2EE server which is used to hold business logic.
12. EJB container provides Local and Remote Access to Enterprise Beans.

[11] SELF-STUDY QUESTIONS

(I) Descriptive Questions :

1. Answer the following questions : **(2 marks)**

(1) Explain N tier Architecture.
(2) Explain Container.
(3) Explain single tier and two tier architecture.
(4) Explain Web Container.

2. Answer the following questions : **(3 marks)**

(1) Explain Enterprise Architecture.

3. Answer the following questions : **(5 marks)**

(1) Explain J2EE API.

(II) Multiple Choice Questions (M.C.Qs) :

1. In this architecture Mainframe Server is used...

(a) Single Tier
(b) Two tier
(c) Three tier
(d) None

2. Application is divided in to Client machine and database server is called _____ architecture.

(a) Three tier
(b) N tier
(c) Two tier
(d) Single tier

3. Which layer is resides at client tier in thin client of two tier architecture ?

(a) Presentation Logic
(b) Presentation Manager
(c) Business Logic
(d) Application Logic

4. **Which layer is not resides at client tier in Thick client of Two tier architecture ?**
   (a) Business Logic (b) Presentation Manager
   (c) Presentation Logic (d) Application Logic

5. **Which layer is not resides at server tier in Normal client of Two tier architecture ?**
   (a) Application Logic (b) Presentation Logic
   (c) Database Logic (d) Business Logic

6. **Business Logic is resides at _____ tier in Three tier architecture.**
   (a) Client tier (b) Business Tier
   (c) Database Tier (d) None

7. **Which protocol is state less protocol ?**
   (a) HTTP (b) HTML
   (c) HTTP Session (d) TCP

8. **Which tier is used to handle SQL request in N-tier architecture ?**
   (a) Client tier (b) Web tier
   (c) Business tier (d) EIS tier

9. **Web tier resides at _____ server in Enterprise architecture.**
   (a) J2EE Server (b) EIS Server
   (c) Database Server (d) None

10. **What executes EJB components ?**
    (a) A web server (b) An application server
    (c) An EJB container (d) A database server

11. **Java Messaging Service (JMS).**
    (a) allows messages to participate in a distributed transaction
    (b) is necessary for sending and receiving e-mails
    (c) is a non-standard Java EE feature of IBM
    (d) None

12. **What is use of JNDI ?**
    (a) Access to different types of Naming and directory services.
    (b) Access to different types of Networking and directory services.
    (c) Access to Directory interfaces.
    (d) None

13. **Which API is used for distributed transaction application ?**
(a) JSF
(b) JMS
(c) JTA
(d) JavaMail

14. **Which API is used for building dynamic web applications ?**
(a) Java Servlet
(b) JSP
(c) JMS
(d) RMI

15. **Which API provides abstraction for building dynamic web applications ?**
(a) Java Servlet
(b) JSP
(c) EJB
(d) JDBC

16. **_____ provides communication platform between client and components.**
(a) Container
(b) J2EE server
(c) Application Server
(d) Database Server

17. **Which is not a type of container ?**
(a) Applet Container
(b) Database Container
(c) Web Container
(d) Application Container

18. **_____ is used to host web applications.**
(a) Web container
(b) EJB container
(c) Applet Container
(d) None

19. **Which is not type of Beans ?**
(a) Entity Beans
(b) Message Driven Beans
(c) Mail Beans
(d) Session Beans

20. **_____ provides local and remote access to Enterprise Beans.**
(a) Web Container
(b) EJB Container
(c) Applet Container
(d) Application Container

ANSWERS

1. (a)	2. (c)	3. (b)	4. (a)	5. (b)	6. (b)	7. (a)	8. (d)	9. (a)	10. (c)
11. (a)	12. (a)	13. (c)	14. (b)	15. (a)	16. (a)	17. (b)	18. (a)	19. (c)	20. (b)

INTRODUCTION TO JDBC

- JDBC stands for Java Database Connectivity.
- The java provides JDBC API to create Java applications that are capable of interacting with a database.
- As we know that Java is an ideal language for persistent data storage.
- By using class inheritance and data encapsulation, a Java application developed to work with one type of data storage can be ported or extended to work with another.
- An example for that is an application currently working with a RDBMS (Relational Database Management System) that is extended to use RMI to store data in a file.
- A general misunderstanding with Java is that database access can possible only with the JDBC API.
- Even though the JDBC provides lower level classes to manage database connections and transactions, it is not a compulsory component.
- We can easily connect our application with Microsoft SQL using MSSQLJAVA package instead of the MSSQLJDBC driver.

Introduction of JDBC :

- JDK 1.0 was released in January 1996.
- It contained all the classes necessary to implement database access-although developers had to code their own connections.
- The JDK contained core classes such as java.net and java.io was grouped together to provide the necessary classes to connect to a database and send-receive data.
- Drawback of these classes for connection is that we have to create a socket and stream data to and form it to implement a simple transaction such like inserting or selecting data from database.
- So overcome from this drawback in February 1996, Sun released the JDBC as a separate package under the name java.sql.
- With the release of JDK 1.1, the JDBC package is not included as part of the core Java classes.
- The JDBC API is based on the SQL CLI (Call Level Interface).
- This standard explains how to access databases with function calls embedded in applications.

- Certain limitations are there when we are using JDBC.
- One of them is to connect to a database the JDBC-ODBC Bridge is used with native method calls and DLL (Dynamic Link Library) connect to a database, so it don't allowed its use in applet accessible through Internet.
- Because of potential security issues of an applet, the JDBC limits network connectivity to same host from which the applet was downloaded.
- Because of potential security risk of an applet, database must be installed on the same server as the HTTP server.

- The following are **characteristics** of JDBC:

1. It is call level SQL interface for Java.
2. It does not restrict the type queries passed to an underlying DBMS driver.
3. JDBC mechanisms are simple to understand and use.
4. It provides a Java Interface that stays consistent with the rest of the Java system.
5. JDBC may be implemented on top of common SQL level APIs.
6. It uses strong, static typing whenever possible.

- Some of the **advantages** of using of Java with JDBC are:

1. Supports a variety of relational Databases.
2. It is easy and economical.
3. It allows continuous usage of already installed databases.
4. It has a short development time.
5. Installation and version control is simple

Functionality of JDBC :

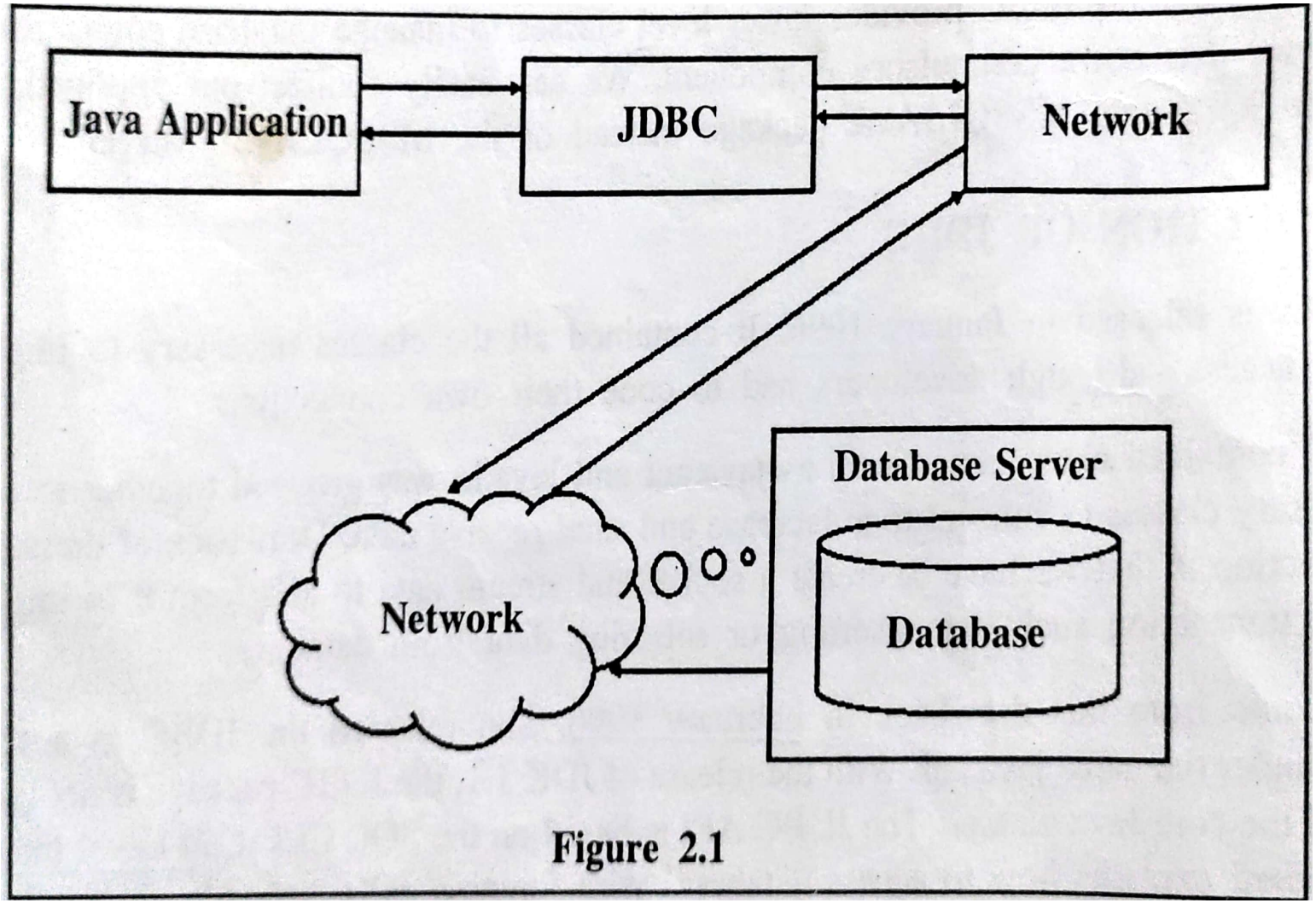


Figure 2.1

- Some of the current trends that are being developed to add more features to JDBC are embedded SQL for Java and direct mapping of relational databases to Java classes.
- Embedded SQL enables mixing of Java into a SQL statement.
- These statements are translated into JDBC calls using SQL processor.
- In this type of direct mapping of relational database tables to Java classes, each row of the table become an instance of the class and each column value corresponds to an attribute of that instance.
- Mapping is being provided that makes rows of multiple tables to form a Java class.

JDBC VS ODBC :

The most widely used interface to access relational database today is Microsoft's ODBC API. ODBC performs similar tasks as that JDBC and yet JDBC is preferred due to following reasons:

1. ODBC can not be directly used with Java because it uses a C interface. Calls from Java to native C code have number of drawbacks on the security, implementation, robustness and automatic portability of application.
2. ODBC makes use of pointers which have been totally removed from Java.
3. ODBC mixes simple advanced features together and has complex options for simple queries. But JDBC is designed to keep things simple while allowing advance capabilities when required.
4. ODBC requires manual installation of the ODBC driver manager and driver on all client machines. JDBC drivers are written in Java and JDBC code is automatically installable, secure and portable on all Java platform from network computers to mainframes.
5. JDBC API is an natural Java interface and is built ODBC JDBC retains some the basic features of ODBC like X/Open SQL call level interface.

Note : A native method is a Java Method, either an instance method or a class method, whose implementation is written in another programming language like 'C'

JDBC Architecture :

- The JDBC architecture describes the interaction of JDBC API with Java Application and Java applet.
- JDBC API consists of several call level interfaces for interaction with JDBC Driver Manager and JDBC drivers for defining databases.
- JDBC API is responsible for transferring data between an application and a database.
- Architecture of JDBC that represents the working of JDBC API, JDBC Driver and JDBC Driver Manager with multiple databases.

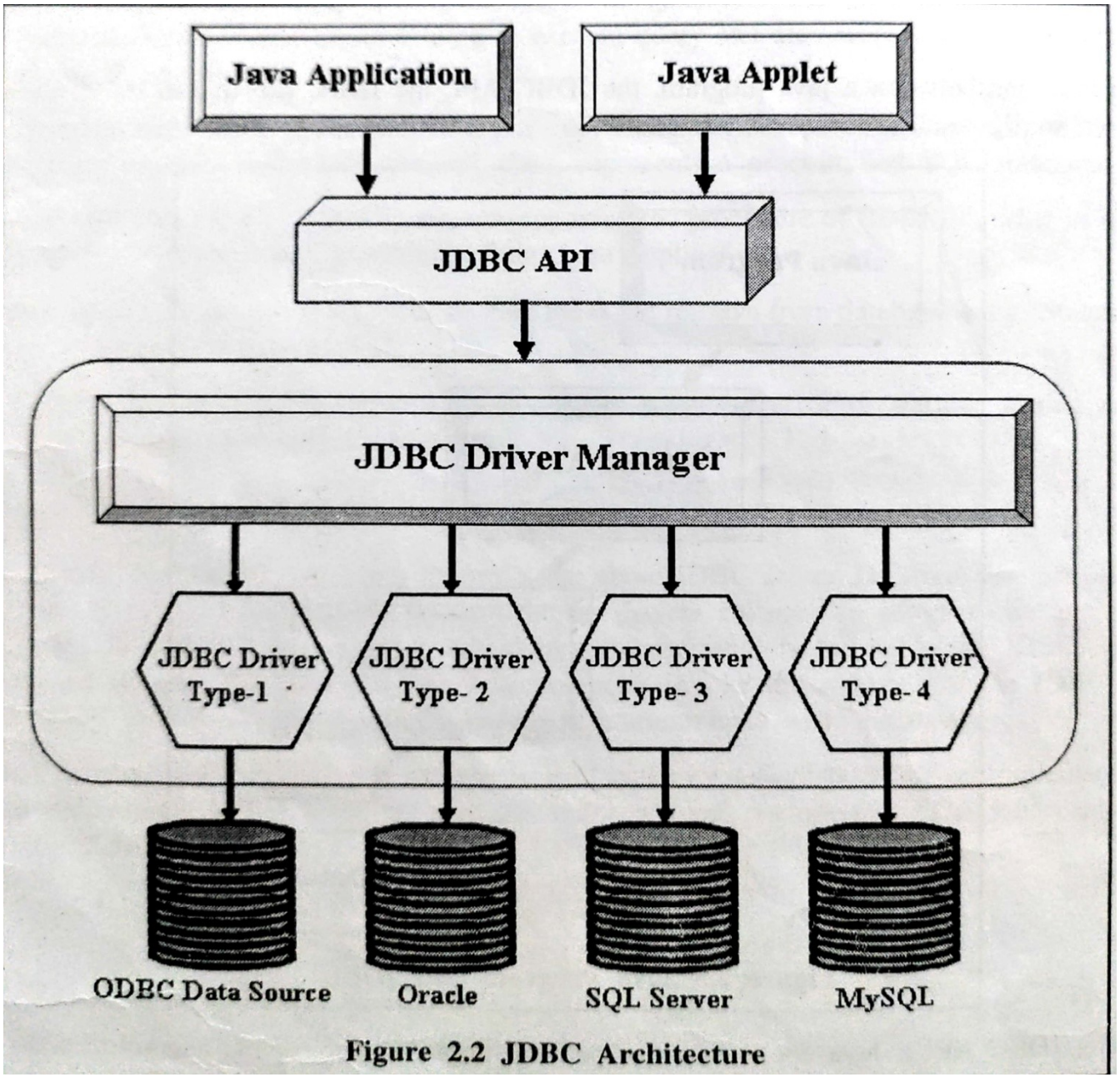


Figure 2.2 JDBC Architecture

The JDBC architecture divided into two parts JDBC Driver Types and JDBC APIs (java.sql and javax.sql packages).
Let us discuss in detail.

JDBC API :

- The JDBC API is a collection of methods, classes and interfaces that enable Java applications to communicate with database.
- For this, database should be installed on the computer and the RDBMS should provide a driver.
- JDBC classes and interfaces defined in the package java.sql constitute the JDBC API The Java program invokes the method of the JDBC APL The JDBC API then calls the JDBC driver and submits the queries to it.
- The JDBC driver then converts the queries to the SQL statements that's database can understand.
- After the query has been performed, the JDBC driver retrieves the result of the query form the database.
- The result is converted into the JDBC API classes that are used by the Java Program.
- Then, the java program obtains the result of the query.
- The relations between a java program, the JDBC API, the JDBC driver, and the database.

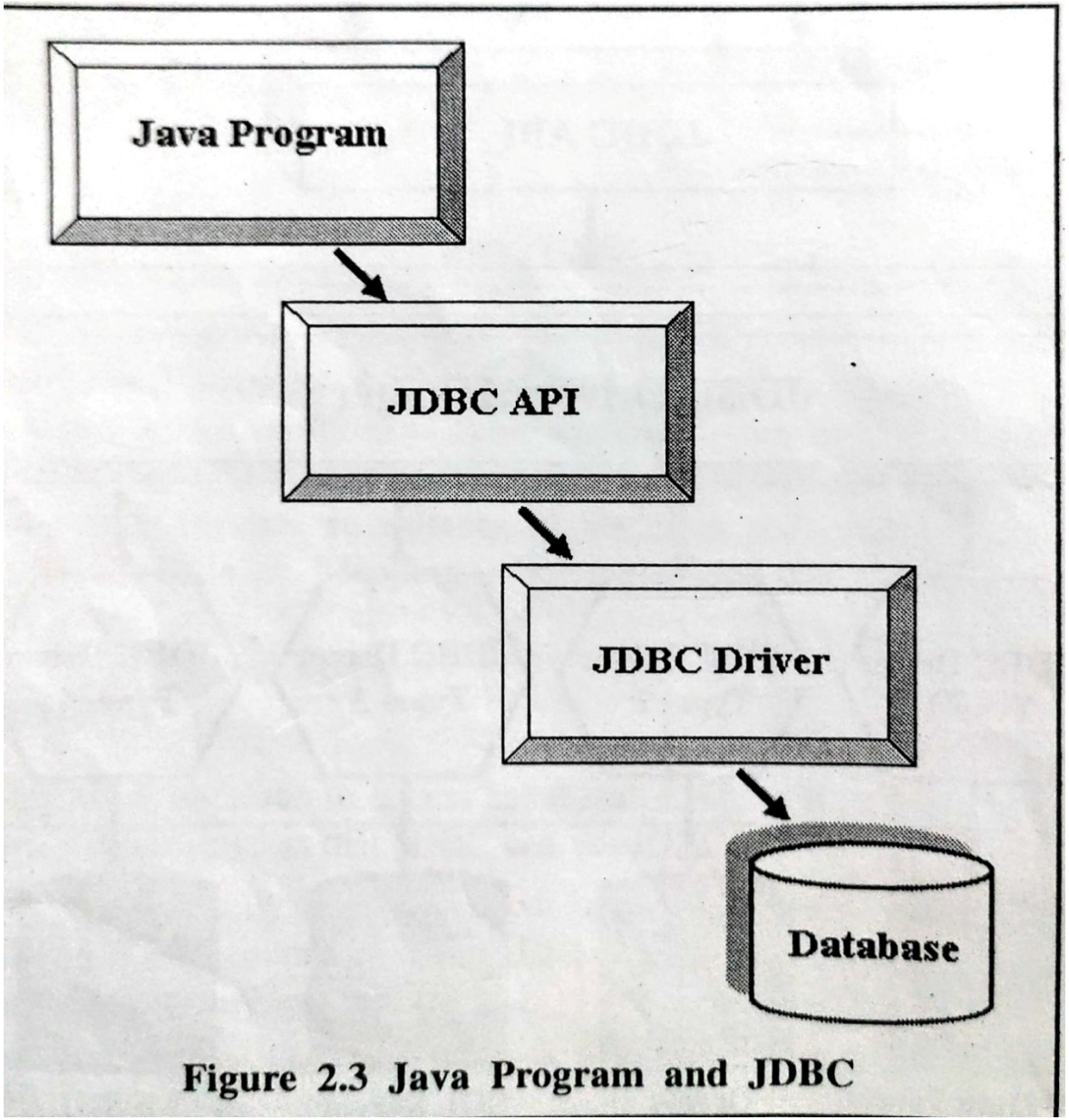


Figure 2.3 Java Program and JDBC

- Using JDBC API a Java application can perform insert, update, delete and select rows in a database.
- It can also retrieve the data from the database and show it in a proper format to the user according to the SQL query.
- The JDBC API can be used in two-tier and three-tier database architectures.
- In two-tier architecture, java programs invoke methods of the JDBC API communicate with the database serve.
- So, there is a direct interaction between Java pangrams and databases.
- In three-tier architecture a Java program submits queries to a server then server uses the JDBC API to interact with the database server.

- Given below some classes and interfaces which support connectivity between interface and database:

1. **DriverManager :** It manages all the Drivers found in JDBC environment, load the most appropriate driver for connectivity.
2. **Connection :** It creates an object which represents connection and its object also helps in creating object of Statement, PreparedStatement and CallableStatement classes.
3. **Statement :** Statement object is used to execute query and also store its value to Resultset object if necessary.
4. **PreparedStatement :** It can be used in place of "Statement", PreparedStatement's performance is high as compared to "Statement" class, represents a precompiled SQL statement.
5. **CallableStatement :** Callablestatement support stored procedure of RDBMS', using its object you can execute stored procedure of database application.
6. **ResultSet :** Resultset object is used to store the result retrieve from database using "Statement" or "PreparedStatement", etc.
7. **SQLException :** SqlException class is used to represent error or warning during access from database or during connectivity.

JDBC Driver Types :

- Before talk on JDBC Driver Types let us discuss about JDBC driver.
- Database vendors provide a driver along with the database.
- These database drivers are used to communicate with Java programs.
- The JDBC drivers are provided by database vendors to enable the JDBC API to communicate with the database.
- The Java programs invoke the methods of the JDBC API.
- JDBC API used the JDBC driver to communicate with the databases.
- JDBC drivers are divided into four categories.
- Each category defines JDBC driver implementation with increasingly higher levels of platform independence, performance.
- The four categories of JDBC drivers are:

1. Type 1: JDBC-ODBC Bridge
2. Type 2: Native-API / partly-Java driver
3. Type 3: Net-protocol / all-Java driver
4. Type 4: Native-protocol/all-Java driver

- **Type 1: JDBC-OBDC Bridge:**

- The JDBC-ODBC bridge driver converts all JDBC calls into ODBC calls and sends them to the ODBC driver.
- The ODBC driver then forwards the call to the database server.
- Figure shows communication between Java application and Database using JDBC-ODBC Bridge.
- For ex. MS Access do not provide JDBC driver.
- Instead they provide Open Database Connectivity (ODBC) drivers.
- One can access such databases by using the OBDC-JDBC Bridge.

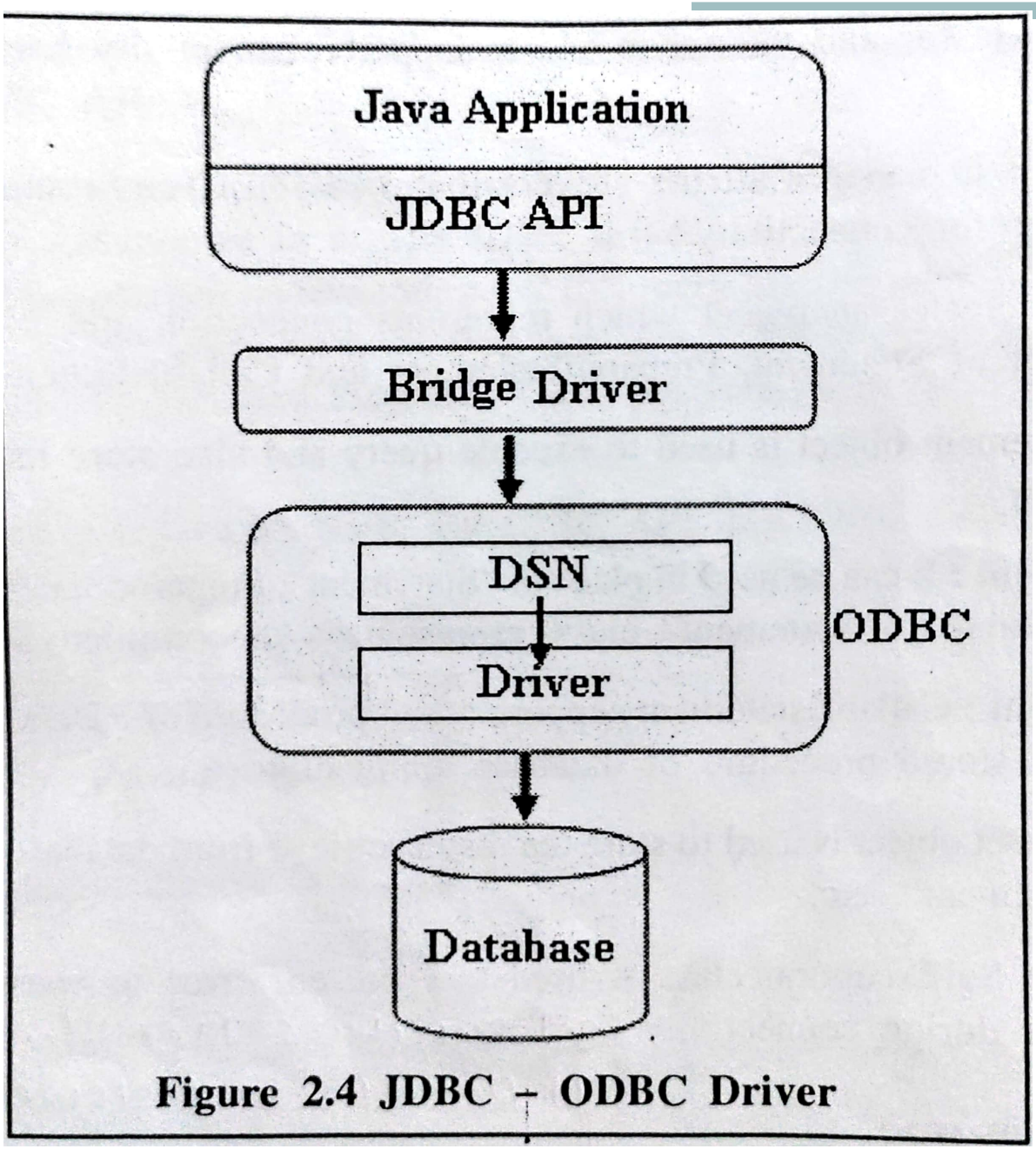


Figure 2.4 JDBC – ODBC Driver

- **Advantage** of the JDBC-ODBC Bridge is it allows access to almost any database, since the database's ODBC drivers are already available.
- **Disadvantages** are:

  1. Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.

  2. Performance is slowest in compare to all drivers because the JDBC calls are first converted to ODBC calls. ODBC calls are then converted to native database calls. The same steps are performed in reverse order when the result of the query is obtained.

  3. The client system requires the ODBC Installation to use the driver.

  4. Not good for the Web Application or for large scale database based applications.

- **Type 2: Native-API/Partly-Java Driver:**

- The JDBC Type 2 driver, or the native-API/partly-Java driver, communicates directly with the database server.
- Therefore, the vendor database library needs to be loaded on each client computer.
- The Type 2 driver converts JDBC calls into database-specific calls for databases.
- Example: Oracle will have oracle Native API.

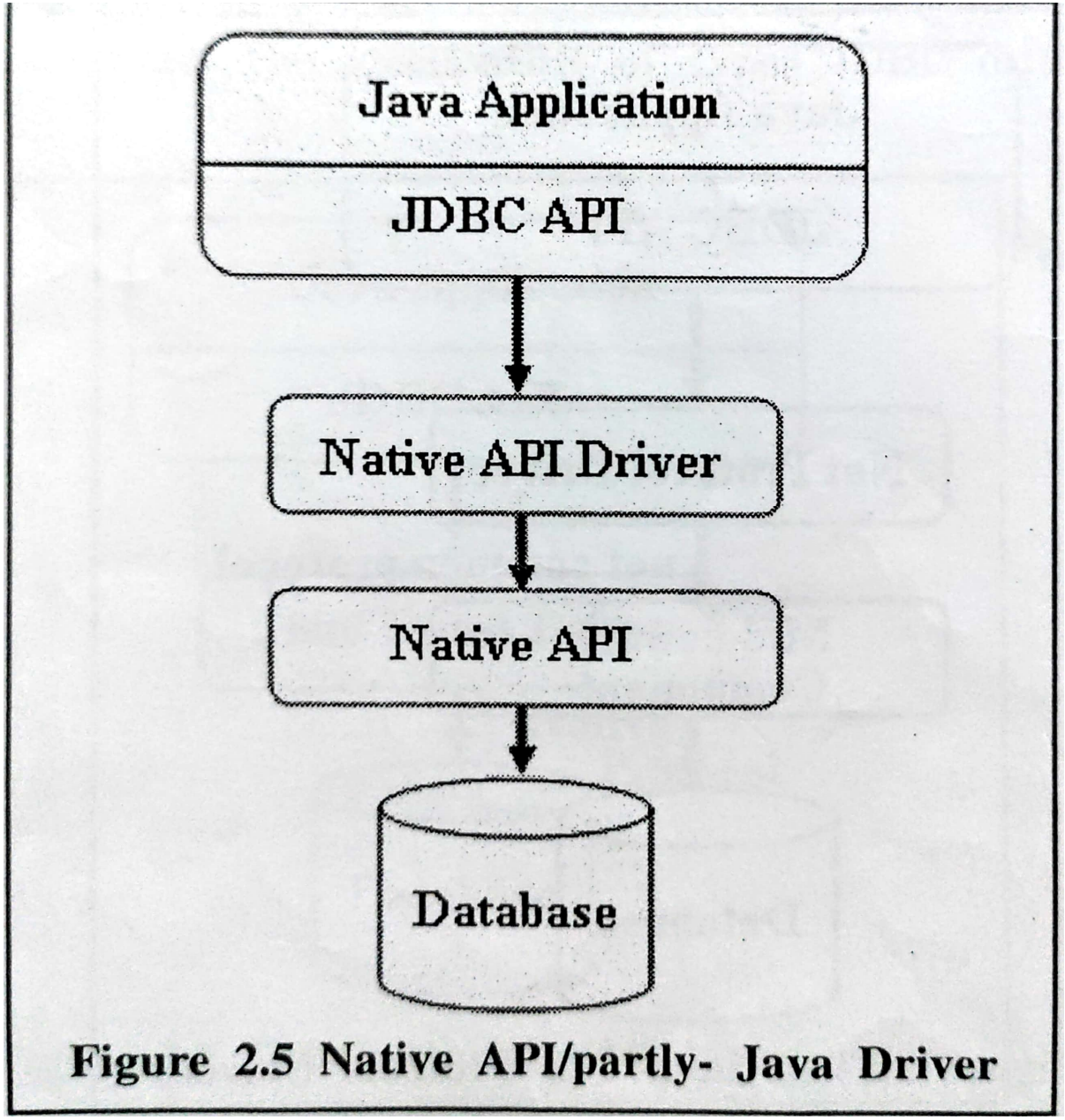


Figure 2.5 Native API/partly- Java Driver

- **Advantage** of Type2 driver are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than and also it uses Native API which is Database specific.
- **Disadvantages** are:

  1. Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.

  2. Like Type 1 drivers, it's not written in Java Language which forms a portability issue.

  3. If we change the Database we have to change the Native API as it is specific to a database.

  4. Mostly outdated now because it is not thread safe.

  5. It is not suitable for distributed application.

- **Type 3: Net-Protocol/All-Java Driver (**Intermediate server driver**) :**

- The JDBC Type 3 driver, or the net-protocol/all-Java driver, follows a three-tiered approach.
- In the three-tiered approach, JDBC database requests are passed to a middle-tier server.
- The middle-tier server then translates the requests are passed to the database-specific native-connectivity interface and forwards the requests to the database server.
- If the middle-tier server is written in java, it can use the Type 1 or type 2 drivers forward the requests to the database server.

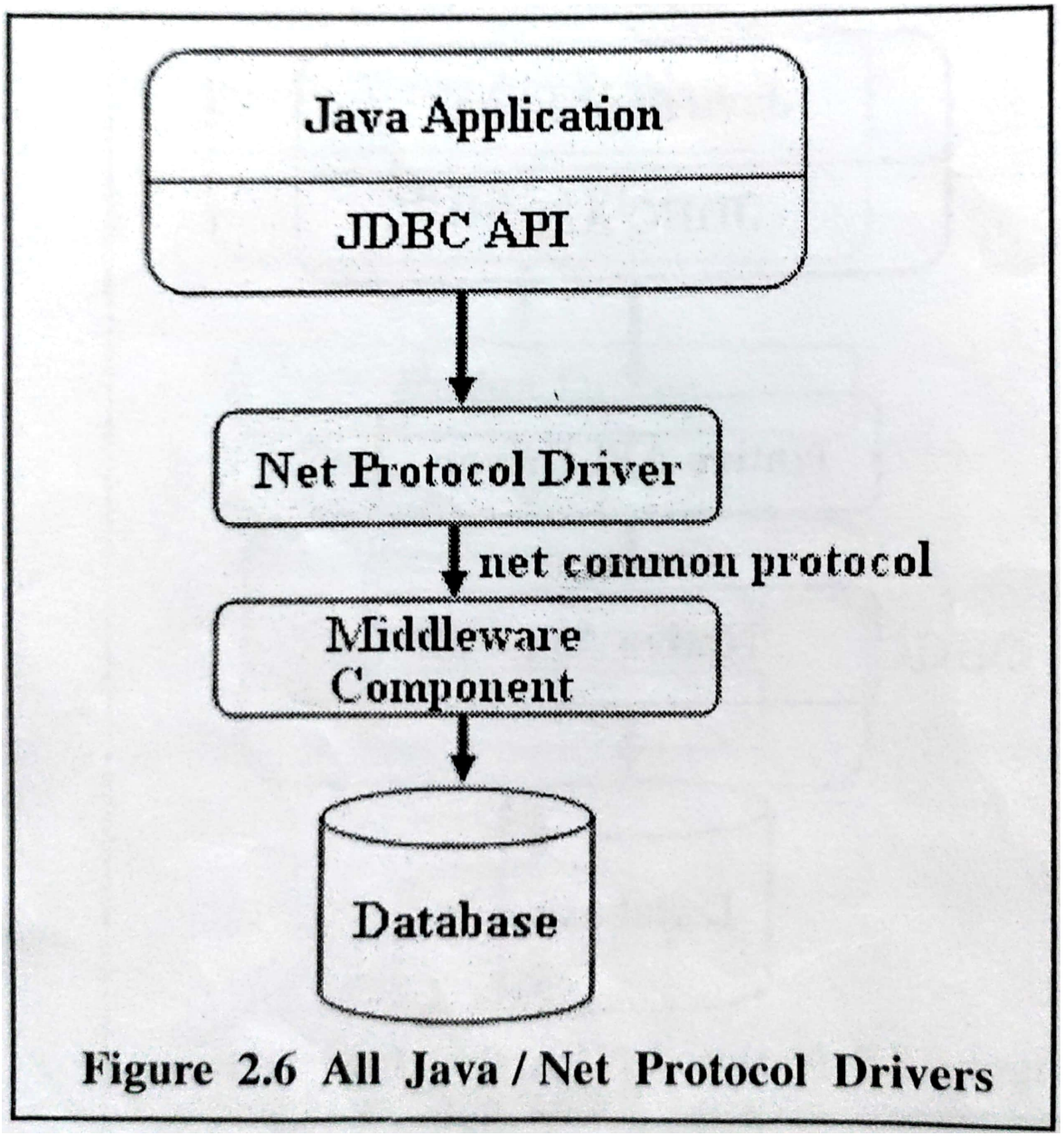


Figure 2.6 All Java / Net Protocol Drivers

- Type 3 drivers are used to connect a client application or applet to a database over TCP/IP connection.
- The presence of any vendor database library on client computers is not required because the Type 3 JDBC driver is server-based.
- The **advantages** of Type3 Driver are:

  1. This driver is server-based, so there is no need for any vendor database library to be present on client machines.

  2. This driver is fully written in Java and hence portable, so it is suitable for the web applications.

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

  4. The net protocol can be designed to make the client JDBC driver very small and fast to load.

  5. The Type3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced system administration such as logging and auditing.

  6. This driver is flexible allows access to multiple databases using one driver.

  7. They are the most efficient amongst all driver types.

- The **disadvantage** of using a Type 3 driver is that it requires database specific coding to be done in the middle tier and it requires additional server for that. Additionally, traversing the recordset may take longer because the data comes through the backend server.

- **Type 4 : Native-Protocol/All-Java Driver (**Pure Java Driver/ Thin Driver**) :**

- Type 4 drivers, or native-protocol/all-Java drivers, are completely implemented in Java to achieve platform independence.
- Type4 JDBC drivers convert JDBC calls into the vendor-specific DBMS protocol.
- Therefore, client applications can communicate directly with the database server when Type 4 JDBC drivers are used.

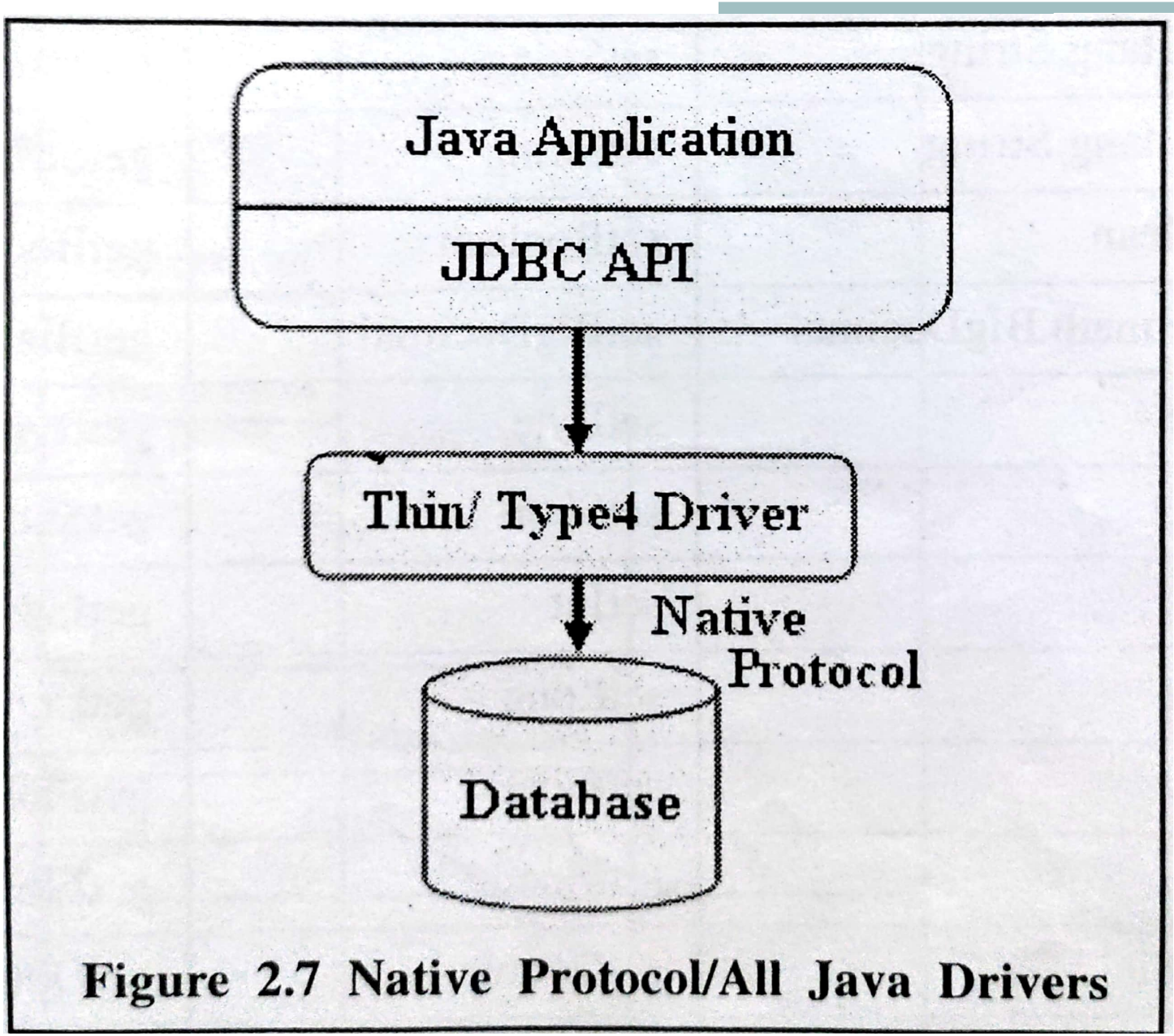


Figure 2.7 Native Protocol/All Java Drivers

- The performance of Type 4 JDBC driver is better than other drivers because Type4 JDBC drivers do not have to translate database requests to ODBC calls or a native connectivity interface or pass the request on to another server.
- A **disadvantage** of Type4 drivers, a different driver is needed for each database.
- Whereas **advantages** are:

  1. The major benefit of using Type4 JDBC drivers is that they are completely written in Java to achieve platform independence And using this benefit we can reduce deployment administration issues. So, it is most suitable for the web application.

  2. Number of translation layers is very less i.e. Type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server. So, performance is typically quite good.

  3. You need not to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Data Types in JDBC :

- The JDBC driver converts the Java data type to the appropriate JDBC type before sending it to the database.
- It uses a default mapping for most data types.
- For example, Java int is converted to an SQLINTEGER.
- Default mappings were created to provide consistency between drivers.
- The following table summarizes the default JDBC data type that the Java data type is converted to when you call the setXXX() method of the PreparedStatement or CallableStatement object or the ResultSet.getXXX () method.

SQL	JDBC / Java	setXXX	getXXX
VARCHAR	java.lang.String	setString	getString
CHAR	java.lang.String	setString	getString
LONGVARCHAR	java.lang.String	setString	getString
BIT	boolean	setBoolean	getBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	getBigDecimal
TINYINT	byte	setByte	getByte
SMALLINT	short	setShort	getShort
INTEGER	int	setInt	getInt
BIGINT	long	setLong	getLong
REAL	float	setFloat	getFloat
FLOAT	float	setFloat	getFloat
DOUBLE	double	setDouble	getDouble
BINARY	byte[ ]	setBytes	getBytes
DATE	java.sql.Date	setDate	getDate
TIME	java.sql.Time	setTime	getTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	getTimestamp
CLOB	java.sql.Clob	setClob	getClob
BLOB	java.sql.Blob	setBlob	getBlob
ARRAY	java.sql.Array	setARRAY	getARRAY
REF	java.sql.Ref	SetRef	getRef
STRUCT	java.sql.Struct	SetStruct	getStruct

Processing Query :

- Following diagram shows java JDBC interaction of the core JDBC components du execution of a database query. Following six steps are there:

1. Load driver.
2. Open and Return connection.
3. Create and Return statement.
4. Execute statement and Return statement if required.
5. Traverse Resultset.
6. Return records.

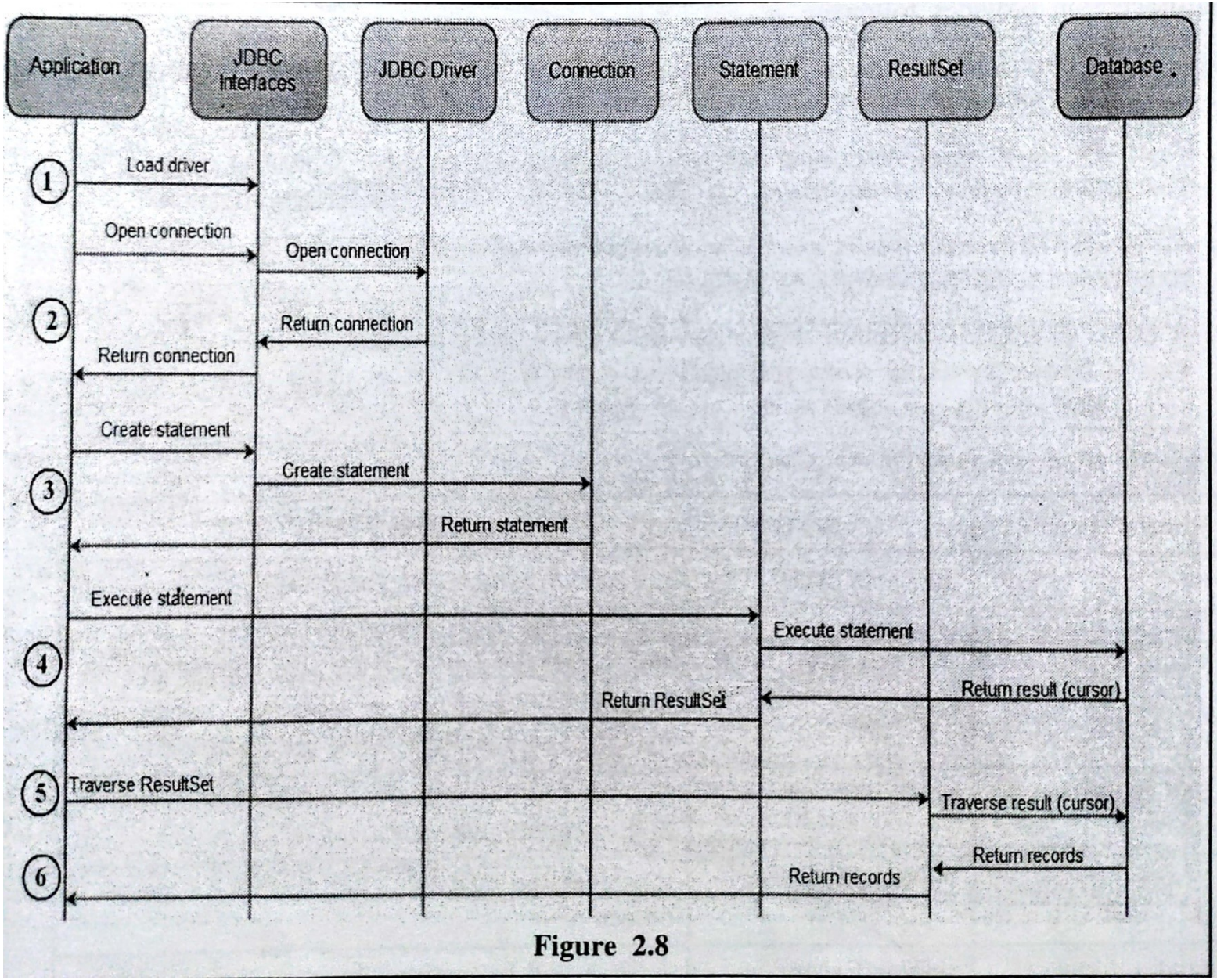

Application
JDBC interfaces
JDBC Driver
Connection
Statement
ResultSet
Database
1
Load driver
Open connection
Open connection
2
Return connection
Return connection
Create statement
Create statement
3
Return statement
Execute statement
Execute statement
4
Return result (cursor)
Return ResultSet
5
Traverse ResultSet
Traverse result (cursor)
Return records
6
Return records

Figure 2.8

Database Exception Handling :

- The SQLException class extends the general java.lang.Exception class.
- It is an exception that provides information on a database access error or other errors.

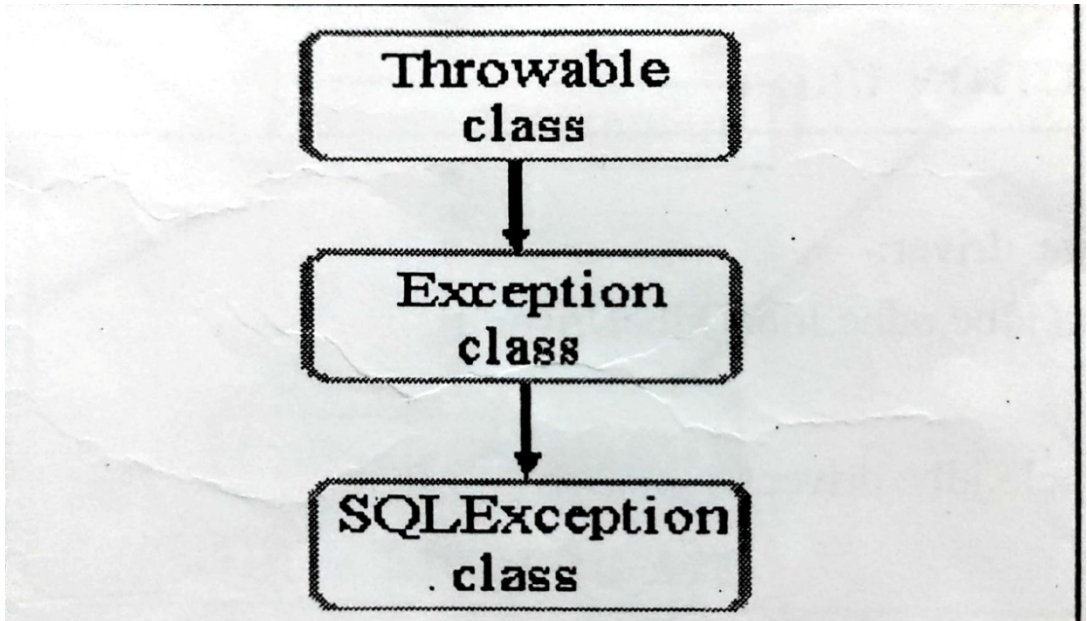


Figure 2.9 Class Hierarchy of SqlException

- SQLException provides following several kinds of information:

1. A string describing the error. This is used as the Java Exception message, available via the method getMessage().
2. The "SQLState" string describing the error according to the XOPEN SQLState conventions. The different values of this string are defined in the XOPEN SQL specification.
3. An integer error code that is specific to each vendor. Normally this will be the actual code returned by the underlying database.
4. A chain to a next Exception, which can be used to provide additional error information. This is frequently useful when you have exceptions that you want to let the user about, but you do not want to stop processing.

Methods of java.sql.SQLException :

Return Type	Method	Description
int	getErrorCode()	It is used to retrieve the vendor-specific exception code for this SQLException object.
String	getMessage()	Gets the JDBC driver's error message for an error handled by the driver or gets the Oracle error number and message for a database error.
SQLException	getNextException()	It is used to retrive the exception chained to this SQLException object.
String	getSQLState()	Retrieves the SQLState for this SQLException object.
void	setNextException (SQLException ex)	Adds a SQLException object to the end of the chain.

Steps for Database Connectivity :

STEPS FOR DATABASE CONNECTIVITY

Following are steps to connect your application to Database.

DEFINE THE CONNECTION URL :

Class.forName();

- For jdbc-odbc bridge driver:-
  Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
- For oracle driver:-
  Class.forName ("oracle.jdbc.driver.OracleDriver");
- For Mysql driver:-
  Class.forName("com.mysql.jdbc.Driver");

* **ESTABLISHED THE CONNECTION :**

- For Oracle or other Sql server:

```
Connection con =DriverManager.getConnection("url","user_name","pass");
```

- For Ms-Access

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

* **CREATE THE STATEMENT OBJECT :**

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

* **EXECUTE A QUERY :**

```
stmt.executeQuery("sql");
                or
stmt.executeUpdate("sql");
```

* **PROCESS THE RESULTS :**

```
ResultSet rs=stmt.execute Query("sql");
while(rs.next( ))
{
      System.out.println(rs.getInt(id));
      System.out.print(rs.getString(name));
}
```

* **CLOSE THE CONNECTION :**

```
stmt.close();
con.close();
```

This figure shows connectivity between all JDBC API which are discussed .

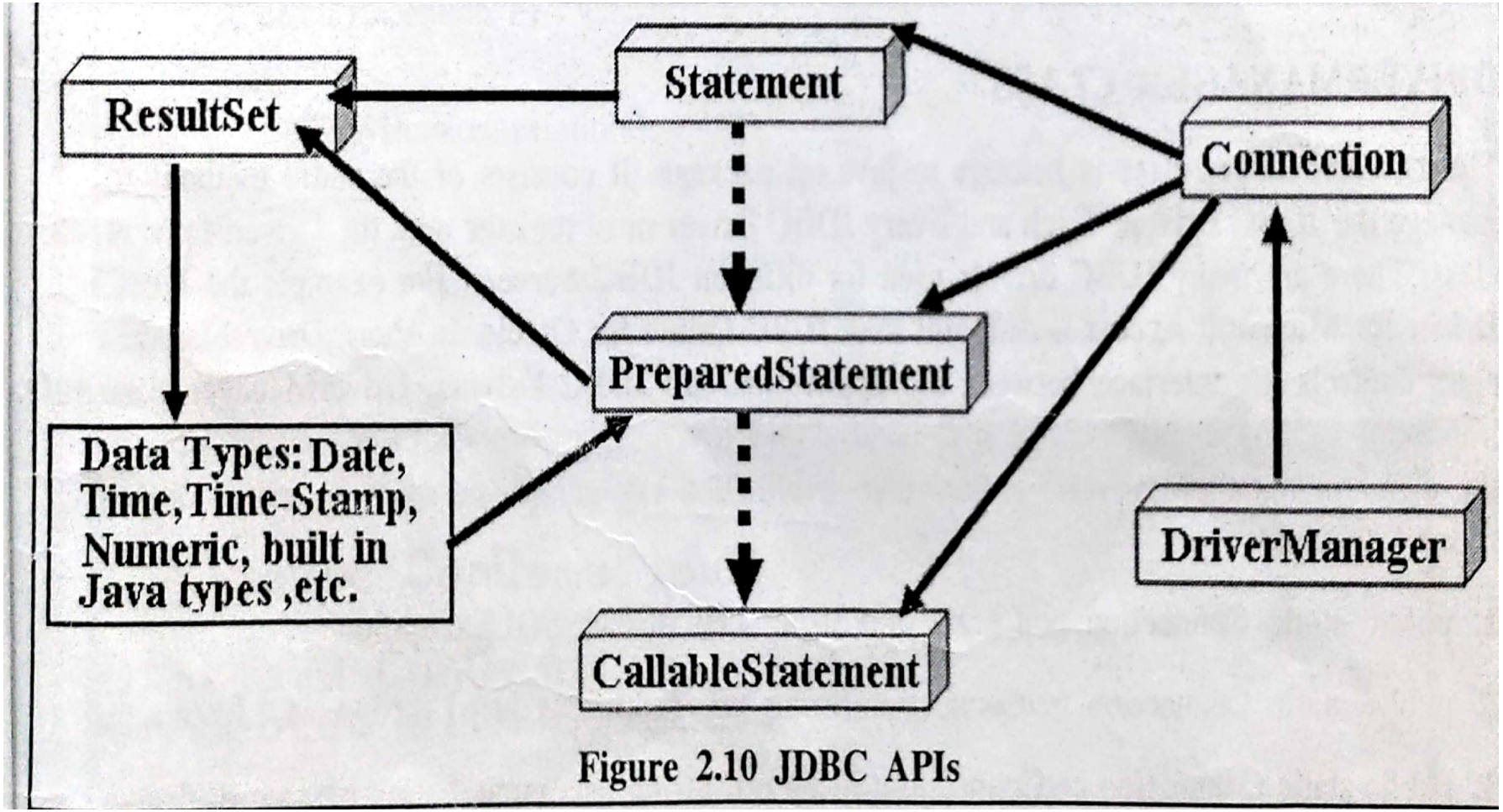


Figure 2.10 JDBC APIs

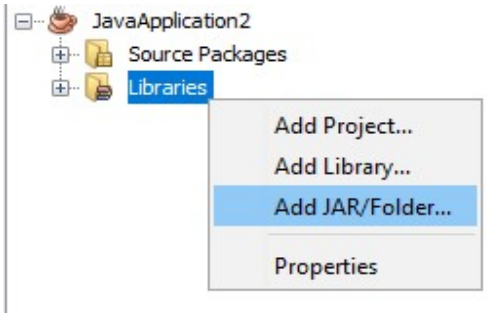
DriverManager Class :

- The DriverManager class is belongs to java.sql package.
- It consists of the static methods to manage the JDBC Drivers.
- Each and Every JDBC Driver must register with the DriverManager class.
- There are many JDBC drivers used for different JDBC Servers.
- For example the JDBC Driver for Microsoft Access is different than JDBC Driver for Oracle.
- In short, DriverManager class controls the interface between the application and JDBC Drivers.
- DriverManager class important method is getConnection ( ) method which is used to establish the connection with the different database servers.
- It has three overloaded signatures (parameters) which are as follows:
- 1. public static Connection getConnection(String url) throws SQLException
- 2. public Connection getConnection(String url, Properties Info) throws SQLException
- **3. public static Connection getConnection(String url, String username, String password) throws SQLExceptiion**

Example :

```
package javaapplication2;
import java.sql.*;

public class JavaApplication2 {
public static void main(String[ ] args) {
try{
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con=DriverManager.getConnection (
                        "jdbc:mysql://localhost:3306/demo" ,"root","");
        System.out.println("Success");
    }
    catch(Exception e)
    {
        System.out.println("Sorry"+e);
    }
  }
}
```



We are used : mysql-connector-java-8.0.20.jar file.

Connection Interface :

- The Connection interface used to connect to a database is available in the java.sql package.
- A Connection object represents an SQL session with a database.
- This interface has methods which can be used to prepare statement.
- This interface also has methods that makes changes to the database permanently or temporary.
- Object of Connection interface :

  Connection con = DriverManager.getConnection(“url”,”user_name”,”password”);

  Statement st=con.createStatement();

Method	Description
void close()	This method frees the Connection object's database and other JDBC resources.
void commit()	This method makes all the changes made since the last commit or rollback permanent.
Statement createStatement()	This method creates a Statement object for sending SQL statements to the database.
boolean isClosed()	This methods returns true if the connection is close else returns false.
CallableStatement prepareCall(String s)	This method creates a CallableStatement object for calling stored procedures.
PreparedStatement prepareStatement (String s)	This method creates a PreparedStatement object for sending SQL statements with or without IN parameter.
void rollback()	This method undoes all changes made to the database.

Statement Interface :

- In this section we will see how the statement object is created for executing static SQL statements.
- To execute SQL statements, statement is the simple and easy.
- The statement interface has several methods which can be used to execute SQL statements.
- 1. executeQuery();
- 2. executeUpdate();
- 3. execute();

- **1. executeQuery():**

  This method is used to fetch the records from the database by using select query.

  It can only return the single ResultSet object at a time.

  It passes a parameter with it which is SQL statement which selects the records from the database.

  Example:

  Statement st=con.createStatement();

  Resultset rs= st.excuteQuery("select from tablename");

- **2. executeUpdate():**

  This methods is used to perform the insert, update and delete operations on the records of the database.

  It returns the integer value which indicates no. of records updated in the database.

  It passed a SQL statement as a parameter which updates the records of the database.

  Example :

  Statement st=con.createStatement();

  int n=st.executeUpdate("intsert into tablename values(vall.val2.....valn)");

- **3. execute() :**

  It is used to execute the SQL statements which return the multiple results.

  It returns the Boolean indicating value if it returns the true then it defines that the next result is ResultSet object and if it returns the false then it defines no. of records updated or no more result is there.

  It passes any SQL statement as parameter.

  Example :

  Statement st=con.createStatement();

  String sql = "create table tablename(fieldname type(size), fieldnametype(size))";

  st.execute(sql);

PreparedStatement Interface :

- PreparedStatement Interface is derived from the Statement Interface.
- Disadvantages of Statement object are that it executes a simple SQL statement with no parameters and the SQL statement is not precompiled.
- But PreparedStatement object uses a template to create a SQL request, and use a PreparedStatement to send precompiled SQL statements with one or more parameters.
- So, sometimes it is more convenient to use a PreparedStatement object for sending SQL statements to the database.
- If you want to execute a SQL statement many times, PreparedStatement object reduces execution time in compare to Statement object.
- The main feature of a PreparedStatement is that SQL statement is given to PreparedStatement object when SQL statement is created.
- So, the advantage of PreparedStatement is that in most of the cases SQL statement is sent to the database immediately, where it is compiled. Using this advantage PreparedStatement object contains not only SQL statement but precompiled SQL statement.

- Though PreparedStatement objects can be used for SQL statements with no parameters, in the majority cases it is used with SQL statements that take parameters.
- Because using SQL statements that take parameters we can use the same statement and supply it with different values each time you execute it.
- **Example :**

  String sql= "insert into tablename (fieldname1, fieldname2) values (?,?)";

  PreparedStatement pstmt = con.prepareStatement(sql);
- **SetXXX Methods:**

  1. setInt(int parameterIndex, int x)
  2. setChar(int parameterIndex, char x)
  3. setString(int parameterIndex, String x)
  4. setFloat(int parameterIndex, float x)
  5. setDouble(int parameterIndex, double x)
  6. setLong(int parameterIndex, long x)
  7. setShort(int parameterIndex, short x)
  8. setDate(int parameterIndex. Date x)
  9. setByte(int parameterIndex, byte x)
  10. setBlob(int parameterIndex, Blob x)

Here, parameter Index the first parameter is 1, the second is 2, and so on x The object containing the input parameter value

- **1. executeQuery() :**

  This method is used to fetch the records from the database by using select query.

  It can only return the single ResultSet object at a time.

  It passes a parameter with it which is SQL statement which selects the records from the database.

  Example :

  String sql= "select *from tablename where fieldname = ?";

  PreparedStatement pstmt = con.prepareStatement (sql);

  pstmt.setString (1,"C01");

  ResultSet rs= pstmt.executeQuery();

- **2. executeUpdate() :**

  This method is used to perform the insert, update and delete operations on the records of the database.

  It returns the integer value which indicates no. of records updated in the database.

  It passed a SQL statement as a parameter which updates the records of the database.

  Example :

  PreparedStatement pst=con.preparedStatement("intsert into tablename values(?,?)");

  pst.setInt(1,101);

  pst.setString(2,”hello”);

  pst.executeUpdate();

- 3. execute():
  It is used to execute the SQL, statements which return the multiple results. It returns the Boolean indicating value if it returns the true then it defines that the next result is ResultSet object and if it returns the false then it defines no. of records updated or It passes any SQL statement as parameter.

CallableStatement Interface :

- To call the stored procedures and functions, CallableStatement interface is used.
- We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.
- A CallableStatement object provides a way to call stored procedures in a standard way for all DBMSs.
- A stored procedure is stored in a database; the call to the stored procedure is what a CallableStatement object contains.
- This call is written in an escape syntax that may take one of two forms: one form with a result parameter, and the other without one.
- A result parameter, a kind of OUT parameter, is the return value for the stored procedure.
- Both forms may have a variable number of parameters used for IN parameters, OUT parameters, or both INOUT parameters.
- A question mark serves as a placeholder for a parameter.
- Syntax for Stored Procedure with IN parameters: (call procedure_name [(?,?, ...)]]
- Syntax for Stored Procedure with IN and OUT parameters:{? call procedure_name [(2, 2, ...)])
- Syntax for Stored Procedure with no parameters{call procedure_name)

- CallableStatement objects are created with the Connection method prepareCall.
- The example below creates an instance of CallableStatement that contains a call to the stored procedure getTestData , which has three arguments and no result parameter.
- CallableStatement cstmt = con.prepareCall("[call getTestData (?, ?))");
- Here? placeholders are may be IN, OUT, or INOUT parameters depend on the stored procedure getTestData.

ResultSet Interface :

- It is used to access the data of the table from the database.
- ResultSet object stores the data of the table by executing the query.
- It also maintains the cursor position for navigation of the data.
- Cursor can move on first, next, previous and last position from the current row position.
- It also fetch the data from the table using getXXX methods depends on column type (getXXX methods means getString(), getInt() etc. methods which is used to fetch the different types of data from table).
- It can fetch the data either passing column name or index number with getXXX methods.
- Index number always starts with one (1).
- **ResultSet Contains Records:**
- A ResultSet consists of records. Each record contains a set of columns. Each record contains the same amount of columns, although not all columns may have a value. A column can have a null value.

- **Creating a ResultSet :**
- As you know that we can create a ResultSet by executing a Statement or PreparedStatement, like this:

  Statement statement=connection.createStatement();

  ResultSet result statement.executeQuery("select *from people");

  Or like this:String sql = "select *from people";

  PreparedStaternent statement = connection.prepareStatement(sql);

  ResultSet result=statement.executeQuery();

  Ierating the ResultSet:

  To iterate the ResultSet use next() method. The next() method returns true if the ResultSet has a next record, and moves the ResultSet to point to the next record. If there were no more records, next() returns false, and you can no longer. Once the next() method has returned false,you should not call it anymore. Doing so may result in an exception. Here is an example of iterating a ResultSet using the next() method:

```
while (result.next())
{
   // get column values from this record
}
```

- **Accessing Column Values:**

  getXXX Methods following all the methods are overloaded methods as we have discussed before we can pass either column name or index number as parameter with them.

  1. getString(): It is used to fetch the String type of records from the table. It returns the String object. It can be written as follow.

  2. getInt(): It is used to fetch the integer types of records from table. It returns integer value. It can be written as follow.

  3. getBoolean(): It is used to fetch the Boolean indicating values from the table. It returns either true or false.

  4. getDouble(): It is used to fetch decimal type of values from the table and returns the double value.

  5. getFloat(): It is used to fetch decimal number.

  6. getDate(): It returns the date type of record from the table.

  7. getLong(): It returns long type of integer values from the table.

  8. getShort(): It returns short type of integer values from the table.

  9. getByete(): It returns bye type of values from the table.

  10. getBlob(): It is used to fetch binary representing large object from the table. It can be image file, audio video files etc.

[16] OTHER APIs (METADATA)

Meaning of Metadata is data about data. There are two types of MetaData interface are available : (1) ResultSet MetaData Interface (2) DatabaseMetaData Interface. Let us discuss both interfaces in detail.

1. ResultSetMetaData Interface :

This interface provides methods for obtaining information about the types and properties of the columns in a Result-Set object. ResultSet Metadata is used to store following information of ResultSet object and table 3.1 describes methods of ResultSet Metadata class.

1. What's the number of columns in the ResultSet?
2. What's a column's name?
3. What's a column's SQL type?
4. What's the column's normal max width in chars?
5. What's the suggested column title for use in printouts and displays?
6. What's a column's number of decimal digits?
7. Does a column's case matter?
8. Is the column a cash value?
9. Will a write on the column definitely succeed?

10. Can you put a NULL in this column?
11. Is a column definitely not writable?
12. Can the column be used in a where clause?
13. Is the column a signed number?
14. Is it possible for a write on the column to succeed? and so on.....

Methods	Description
getCatalogName(int column)	Gets the designated column's table's catalog name.
getColumnClassName (int column)	Returns the fully-qualified name of the Java class whose instances are manufactured if the method ResultSet.getObject is called to retrieve a value from the column.
getColumnCount()	Returns the number of columns in this ResultSet object.
getColumnDisplaySize(int column)	Indicates the designated column's normal maximum width in characters.
getColumnLabel(int column)	Gets the designated column's suggested title for use in printouts and displays.
getColumnName(int column)	Get the designated column's name.
getColumnType(int column)	Retrieves the designated column's SQL type.
getColumnTypeName(int column)	Retrieves the designated column's database-specific type name.
getPrecision(int column)	Get the designated column's number of decimal digits.
getScale(int column)	Gets the designated column's number of digits to right of the decimal point.
getSchemaName(int column)	Get the designated column's table's schema.
getTableName(int column)	Gets the designated column's table name.
isAutoIncrement(int column)	Indicates whether the designated column is automatically numbered, thus read-only.
isCaseSensitive(int column)	Indicates whether a column's case matters.
isCurrency(int column)	Indicates whether the designated column is a cash value.
isDefinitelyWritable(int column)	Indicates whether a write on the designated column will definitely succeed.
isNullable(int column)	Indicates the null ability of values in the designated column.
isReadOnly(int column)	Indicates whether the designated column is definitely not writable.

- **Example of ResultSetMetaData :**

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

Connection con=
   DriverManager.getConnection("jdbc:mysql://localhost:3306/demo",
   "root","");

Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select *from stud");

ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("1st Column name :"+ rsmd.getColumnName(1));
System.out.println("2nd Column name :"+ rsmd.getColumnName(2));
System.out.println("Total Column:" +rsmd.getColumnCount());
System.out.println(“1st Column datatype Class name :"+
   rsmd.getColumnClassName(1));
```

2. DatabaseMetatData Interface :

This interface describes the database as a whole. Many methods of the DataBaseMetadata interface return lists of information in the form of ResultSet objects. This interface provides many methods that represent comprehensive information of the database. Database Metadata is used to store following information of Database Metadata object and table 3.2 describes methods of Database Metadata class.

1. What tables are available?
2. What's our user name as known to the database?
3. Is the database in read-only mode?
4. If table correlation names are supported, are they restricted to be different from the names of the tables? And so on...

Methods	Description
allProceduresAreCallable( )	Retrieves whether the current user can call all the procedures returned by the method getProcedures.
allTablesAreSelectable( )	Retrieves whether the current user can use all the tables returned by the method getTable in a SELECT statement.

getCatalogs( )	Retrieves the catalog names available in this database.
getCatalogSeparator( )	Retrieves the String that this database uses as the separator between a catalog and table name.
getConnection( )	Retrieves the connection that produced this metadata object.
getDatabaseProductName( )	Retrieves the name of this database product.
getDatabaseProductVersion( )	Retrieves the version number of this database product.
getDriverName( )	Retrieves the name of this JDBC driver.
getDriverVersion( )	Retrieves the version number of this JDBC driver as a String.
getMaxConnections()	Retrieves the maximum number of concurrent connections to this database that are possible.
getMaxRowSize( )	Retrieves the maximum number of bytes this database allows in a single row. 4052 Default size for character
getURL( )	Retrieves the URL for this DBMS.
getUserName( )	Retrieves the user name as known to this database.
isReadOnly( )	Retrieves whether this database is in read-only mode.
supportsTransactions( )	Retrieves whether this database supports transactions.
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	Retrieves a description of the stored procedures available in the given catalog.

- **Example of DatabaseMetaData :**

```
Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/demo",
"root","");
        DatabaseMetaData dmd=con.getMetaData();
        System.out.println("Driver Name : "+dmd.getDriverName());
        System.out.println("Driver version : "+dmd.getDriverVersion());
        System.out.println("Name of URL : "+dmd.getURL());
        System.out.println("User Name : "+dmd.getUserName());
```

[19] SUMMARY

1. JDBC = **Java Database Connectivity**.
2. Java provides JDBC API to create Java applications for application – database communication.
3. The JDBC API = methods + classes + interfaces that enable Java applications to communicate with database.
4. Using JDBC API a Java application can perform insert, update, delete and select rows in a database.
5. The JDBC architecture describes the interaction of JDBC API with Java Application and Java applet.
6. The JDBC architecture divided into two parts JDBC Driver Types and JDBC APIs.
7. The four categories of JDBC drivers are : Type 1 : JDBC-ODBC Bridge, Type 2 : Native-API / partly-Java driver, Type 3 : Net-protocol / all-Java driver, Type 4 : Native-protocol/ all-Java driver.
8. JDBC API's are used to perform different operations with the database such as.
9. Connection interface is used to establish the connection.
10. DriverManager class manages the different types of Database drivers.
11. Statement and PreparedStateent interface is used to send SQL statements to the database and then execute it.
12. CallableStatement is used to execute the Stored procedure with application.
13. ResultsetMetadta interface gives the metadata related to the table which with current Resultset.
14. Databasemetata give metadata related to the database driver which is the current connection.
15. SQLEXception used to find the runtime errors which occurs when any problem with SQL sttemenets or drivernames.

[20] SELF-STUDY QUESTIONS

(I) Descriptive Questions :

1. Answer the following questions : **(2 marks)**

(1) Explain basic JDBC steps for Database Connectivity.

(2) Explain JDBC Driver Types.

(3) How to process queries in JDBC.

2. **Answer the following questions :** **(3 marks)**

(1) Explain 3 methods of statement interface to execute sql queries.

(2) Explain DatabaseMetaData interface.

(3) Explain ResultSetMetaData interface.

(4) Explain ResultSet interface.

(5) State difference between Statement interface and PreparedStatement interface.

(6) Explain Native protocol driver.

(7) Explain Statement interface.

3. **Answer the following questions :** **(5 marks)**

(1) Explain JDBC driver with its type.

(2) Explain Connection Interface.

(3) Explain steps for connectivity with example.

4. **Answer the following questions :**

(1) Explain different types of JDBC driver managers. **(S. U. December 2014)**

(2) Write a program using RMI and JDBC using stored procedure. **(S. U. December 2014)**

(3) Explain ResultserMetaData and DatabaseMetaData. **(S. U. December 2014)**

(4) Explain difference between Statement and PreparedStatement interface. Write a program to update data in to database using Prepared statement and type-1 driver. **(S. U. December 2014)**

(II) Multiple Choice Questions (M.C.Qs) :

1. **JDBC stands for...**

(a) Java Data Connection
(b) Java Database Connectivity
(c) Java Direct Connect
(d) Java Database Communication

2. **In which version of jdk JDBC was introduced...**

(a) 1.0
(b) 1.1
(c) 1.2
(d) 1.7

3. **Which packages contain the JDBC classes ?**

(a) java.sql
(b) java.rmi
(c) java.lang
(d) java.io

4. **OBDC stands for...**
   (a) Object Data connection
   (b) Object Database Connectivity
   (c) Open Data Connectivity
   (d) Open Database Connectivity

5. **DSN stands for...**
   (a) Data Source Name
   (b) Database Select Name
   (c) Database Source Name
   (d) Data Select Name

6. **What do you mean by Datasource in JDBC ?**
   (a) A DataSource is the basic service for managing a set of JDBC drivers.
   (b) A DataSource is the Java representation of a physical data source.
   (c) A DataSource is a registry point for JNDI-services.
   (d) A DataSource is a factory of connections to a physical data source.

7. **Which driver is mostly used with middle-ware components ?**
   (a) Type1 Driver (JDBC–ODBC Bridge)
   (b) Type 2 Driver (Native API)
   (c) Type3 Driver (Net Protocol)
   (d) Type4 Driver (Native Protocol)

8. **Which is not the method of Connection interface ?**
   (a) close( )
   (b) getMetaData( )
   (c) getConnection( )
   (d) commit( )

9. **Which URL can't return connection object ?**
   (a) con=DriverManager.getConnection( );
   (b) con=DriverManager.getConnection(url);
   (c) con=DriverManager.getConnection(url, Properties Info);
   (d) con=DriverManager.getConnection(url,"username","password");

10. **Which method is used to send precompiled query to the database ?**
    (a) createStatement( )
    (b) prepareStatement( )
    (c) prepareCall( )
    (d) execute( )

11. **Which method is used to call stored procedure from the database ?**
    (a) createStatement( )
    (b) prepareStatement( )
    (c) prepareCall( )
    (d) execute( )

12. **What is the use of Class.forName( ) ?**

(a) Send the sql statement to the database

(b) Set connection with database

(c) Set the driver based on JDBC Drivers

(d) None of above

13. **Which method is used to select the records from the table ?**

(a) executeUpdate( ) (b) execute( )

(c) executeQuery( ) (d) all of above

14. **Which interface used to get information related to connected table of the database ?**

(a) DatabaseMetaData (b) ResultsetMetaData

(c) Resultset (d) Connection

15. **To fetch the binary reprinting data like image, video etc. Which method can be used ?**

(a) getByte( ) (b) getBlob( )

(c) getDate( ) (d) getBoolean( )

16. **Which class of interface contains transaction methods like setAutoCommit, Commit and Rollback ?**

(a) Connection (b) ResultSet

(c) Statement (d) DriverManager

17. **This method is used for retrieving a string value (SQL type VARCHAR) and assigning into java String object.**

(a) getVarchar( ) (b) getObject( )

(c) getString( ) (d) None

18. **A ResultSet object contains a set of rows from a result set or some other source of tabular data, like a file or spreadsheet.**

(a) True (b) False

ANSWERS

1. (b)	2. (a)	3. (a)	4. (d)	5. (a)	6. (d)	7. (c)	8. (c)	9. (a)	10. (b)
11. (c)	12. (c)	13. (c)	14. (b)	15. (b)	16. (a)	17. (c)	18. (a)		

✡ ✡ ✡ ✡ ✡

- J2EE : Java 2 platform Enterprise Edition
- JSF : Java Server Faces
- JMS : Java Message Service
- EJB : Enterprise JavaBeans
- EIS : Enterprise Information Systems
- API : Application Programming Interface
- JDK : java Development Kit
- JRE : Java Runtime Environment
- JNDI : Java Naming and directory Interface
- RMI : Remote Method Invocation
- MIME : Multipurpose Internet Mail Extensions
- DSN : Data Source Name

IMP Questions :

- History of J2EE.
- Driver Types.
- Java.sql package
- J2EE API.
- ResultSet
- Types of MetaData(ResultSetMetadata, DatabaseMetaData)
- Types of Container
- Three-tier & N-Tier Architecture
- CallableStatement and PreparedStatement
- ececuteUpdate()
- Write a JDBC Program to insert and fetch records from database.