Types Of Applications

- Applications are categorized into four types:

  - Standalone Applications
  - Web Applications
  - Enterprise Applications
  - Mobile Applications

Standalone Applications

- Standalone applications, also known as desktop applications, are installed and run on a single machine. They perform their tasks locally without needing an internet connection.
- These applications are categorized based on the type of user interface:

CUI Applications (Console User Interface):
These applications interact with the user via the command line or terminal (Console). Users type commands, and the application responds in text format.

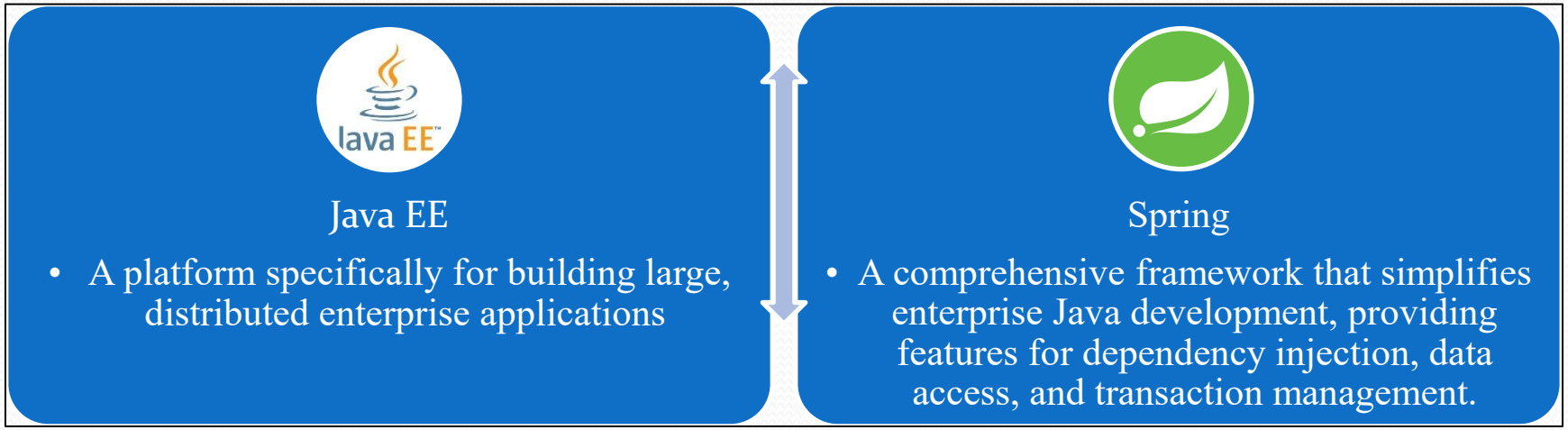
GUI Applications (Graphical User Interface):
GUI applications interact with users through graphical elements like buttons, text fields, and windows.

Web Applications

- Web applications run in a web environment and are accessed through a web browser. They typically consist of a **client-side interface** and a **server-side component** that handles business logic and database interactions.
- In-order to develop these Applications with the help of java we need to have an idea about

Enterprise Applications:

- Enterprise applications are large-scale distributed applications that support complex business processes and high scalability. These applications often require features such as **security**, **load balancing**, **clustering**, and **transaction management**.
- Java is a popular language for building enterprise-level applications due to its robustness and scalability. Key technologies and frameworks used include:



Mobile Applications:

- Mobile applications run on smartphones or tablets, typically designed for iOS or Android platforms. Java is commonly used to develop Android applications because Android is built on the Java platform.
- Java has been the primary language for developing Android applications, utilizing the Android SDK (Software Development Kit)

Types of Storages in Java

Field Storage

- Field storage in Java typically refers to storing data directly in class fields (variables). This is useful for keeping simple, temporary data within an object

Object Storage

- Object storage in Java involves serializing objects to store them persistently. This can be achieved using Java Serialization. This used to store entire Objects in to a file

File Storage

- File storage in Java refers to reading from and writing into files. (IO-Streams). This is used to store text files, logs, and binary data. It is good for small to medium-sized data.

Database Storage

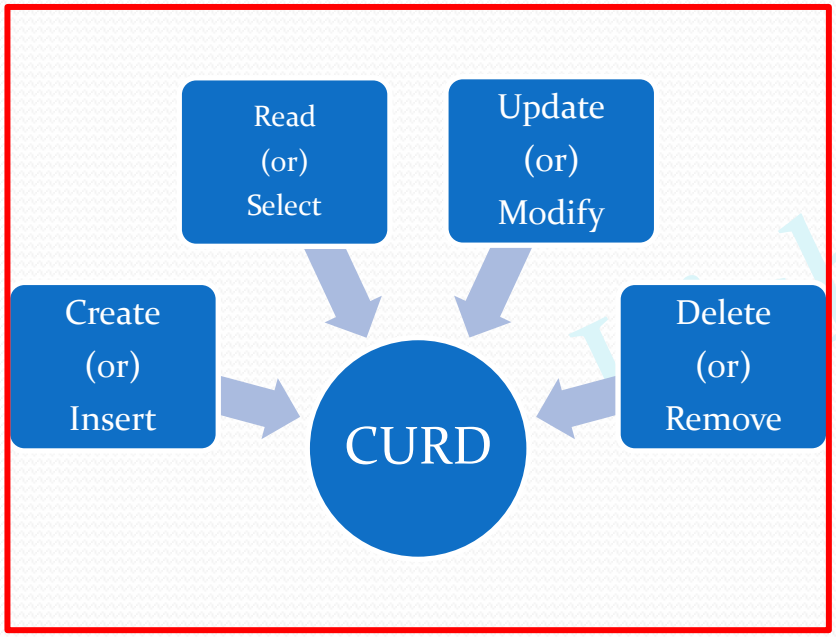
- Database storage involves using JDBC (Java Database Connectivity) to interact with relational databases like MySQL, Oracle, etc. It is used to store large amounts of structured data, supporting complex queries, transactions, and relationships between data.

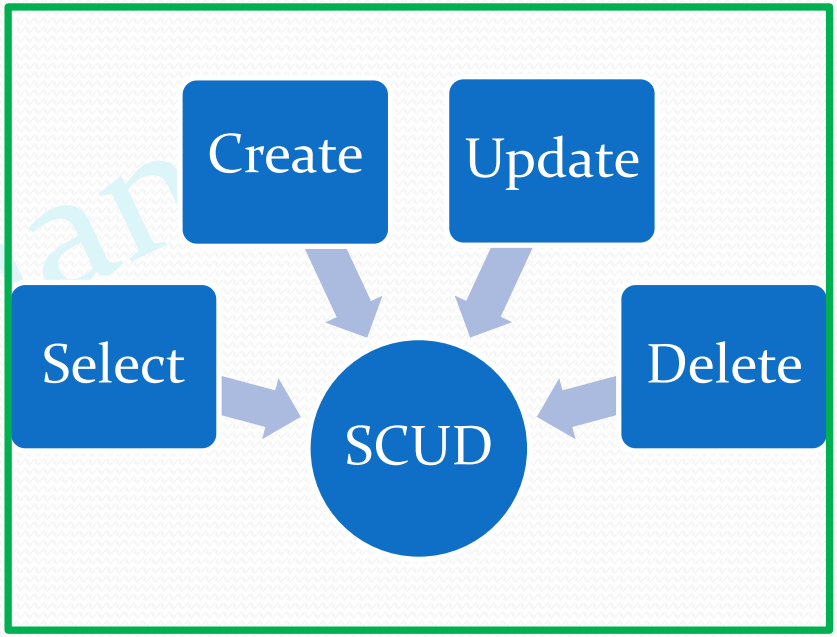
JDBC

Understanding the Terminologies

- **Persistence** : It is a process of saving and managing data for long time. (even after the application that created it has been terminated)
- **Persistence store:** The place where data will be saved and managed ensuring it remains available even after the system is turned off or restarted called persistence store. eg: Files, Database software.
- **Persistent Data** : It refers to information that is stored in a durable storage medium, such as a databases & file system and is preserved beyond the runtime of the application that created it. (allowing it to be retrieved and reused over long periods)

- **Persistence operations:** Insert, update, delete and select operations are persistence operations. (CURD/CRUD/SCUD operations)





- **Persistence logic**: The logic that can be used to perform persistence operations is called persistence logic. eg: I/O streams code, **JDBC**

JDBC Introduction

- **Early Days of Databases:**

Various database vendors developed specific database products. Programmers needed complete knowledge of the native library (often written in 'C') associated with each database. Dealing with databases was complex due to the workings of these native libraries.

- **ODBC (Open Database Connectivity):**

Database vendors collaborated and developed XOPEN/CLI (Call Level Interface) software along with Microsoft which is known as ODBC (Open Database Connectivity). ODBC provided a common API or library for various databases. It was platform-dependent.

JDBC (Java Database Connectivity)

SUN Microsystems developed JDBC.JDBC offered a common API for all databases. It was platform-independent, making it easier for Java applications to interact with databases.

Feature	ODBC (Open Database Connectivity)	JDBC (Java Database Connectivity)
Introduced by	Microsoft (1992)	SUN Microsystems (1997)
Usage	General API for various languages	Java-specific interface
Platform	Primarily Windows	Any platform
Language	Native languages (C, C++)	Java
Paradigm	Procedural	Object-oriented

JDBC architecture

JDBC is like a bridge that helps the Java program talk to a database. It's architecture is made up of different parts that work together which are given below

1. **JDBC API:** Provides interfaces and classes for connecting to and interacting with a database.
2. **JDBC Driver Manager:** It is a Class which manages database drivers and establishes connections.
3. **JDBC Driver:** Translates JDBC calls into database-specific calls.(There are '4' JDBC Drivers)
4. **Connection:** It is an Interface which represents a connection to a database.
5. **Statement:** It is an Interface which is used to execute SQL queries. (There are '3' Statements)
6. **ResultSet:** It is Interface which represents the result set of a query.
7. **SQLException**: It is a Class which handles database access errors and exceptions

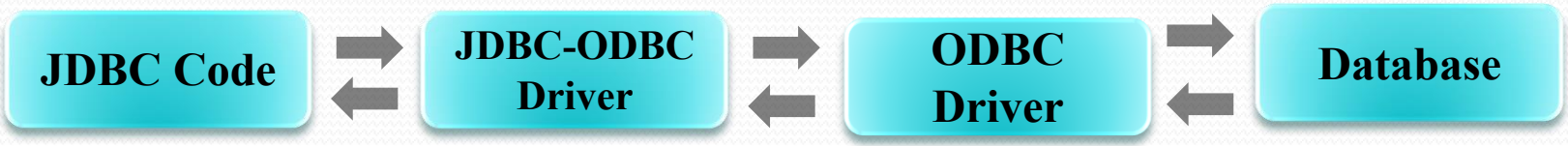
JDBC drivers

The JDBC API uses drivers to connect to different databases. These drivers serve as a bridge between a Java application and the database, converting Java calls into database-specific calls. There are four main types of JDBC drivers, each designed to meet different needs and use cases.

- JDBC-ODBC Bridge driver(Type-1)
- Native API Driver(Type-2)
- Network Protocol Driver(Type-3)
- Thin Driver(Type-4)

JDBC-ODBC Bridge driver(Type-1)

ClassName : sun.jdbc.odbc.JdbcOdbcDriver



- It is developed by SUN micro systems.
- It translates JDBC calls into ODBC Driver calls.
- It acts as a bridge between JDBC code & ODBC Driver.

Advantages:

1. Type-1 driver is by default available in JDK(Up to Java 1.7)
2. It can be used to interact multiple Databases
3. It is Database independent.

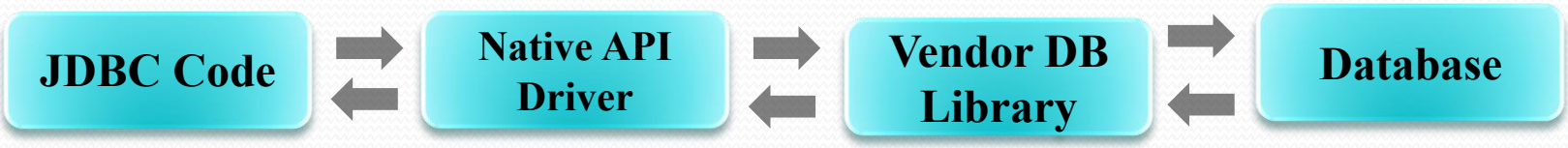
Drawbacks:

1. It is SLOW driver
2. Its performance is not suitable for real time projects.
3. This driver is removed from JDK 1.8 onwards

Native API Driver(Type-2)

ClassName : oracle.jdbc.driver.OracleDriver

(Specific to the database)



- It is developed Database vendors (e.g., Oracle, Sybase, etc.)
- It translates JDBC calls into native API calls specific to the database.
- It acts as a bridge between JDBC code & native database API

Advantages:

1. Faster than Type-1
2. Offers better performance than Type-1, as there's only one translation (JDBC to native API).

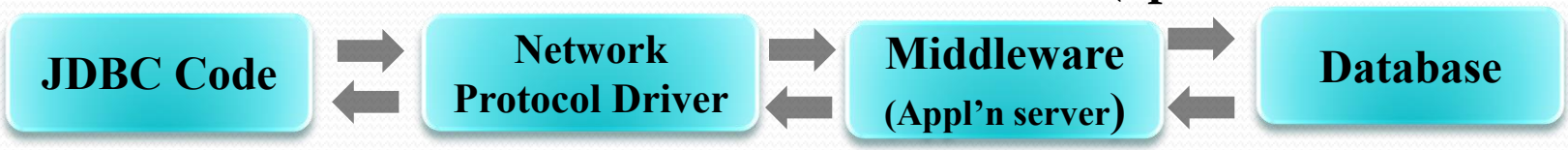
Drawbacks:

1. Requires native libraries to be installed
2. Not fully platform-independent

Network Protocol Driver (Type-3)

ClassName : com.sybase.jdbc3.jdbc.SybDriver for Sybase

(Specific to the Vendor)



- It is developed Database middleware providers
- It translates JDBC calls into a database-independent network protocol which are then translated into database-specific calls by a middleware server.
- It acts as a bridge between JDBC code & and a middleware server

Advantages:

1. Best suited for web applications because it reduces the load on client machine
2. Provides better performance over networks

Drawbacks:

1. Requires an additional middleware server
2. Slightly slower than Type-4 (Due to Complex setup)

Thin Driver (Type-4)

ClassName : oracle.jdbc.OracleDriver **(Database Specific)**

- It is developed Database vendors
- These drivers are written entirely in Java and communicate directly with the database using the database's native protocol
- Direct JDBC connection to the database without additional translation layers
- Advantages:

1. Pure Java driver
2. Provides better performance because it directly interacts with the database
3. Most widely used in modern web and enterprise applications.

Drawbacks:

1. Limited to a single database

Steps for Preparing the database

- Download & Install Database (Oracle / Mysql)
- Connect to the Database
- Create Employee table as given below

Eid	EFname	ELname	ESal	EAddress
101	Kishan	Basina	10000	Hyd
xxx	xxx	xxx	xxx	xxx
xxx	xxx	xxx	xxx	xxx
xxx	xxx	xxx	xxx	xxx
xxx	xxx	xxx	xxx	xxx

- Make Eid as Primary Key
- Copy Oracle jar from the given below location in to user defined folder in the desktop

C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib

- Make a note of Port Number & Service name from the below location (Port No : 1521 & ServiceName : XE)

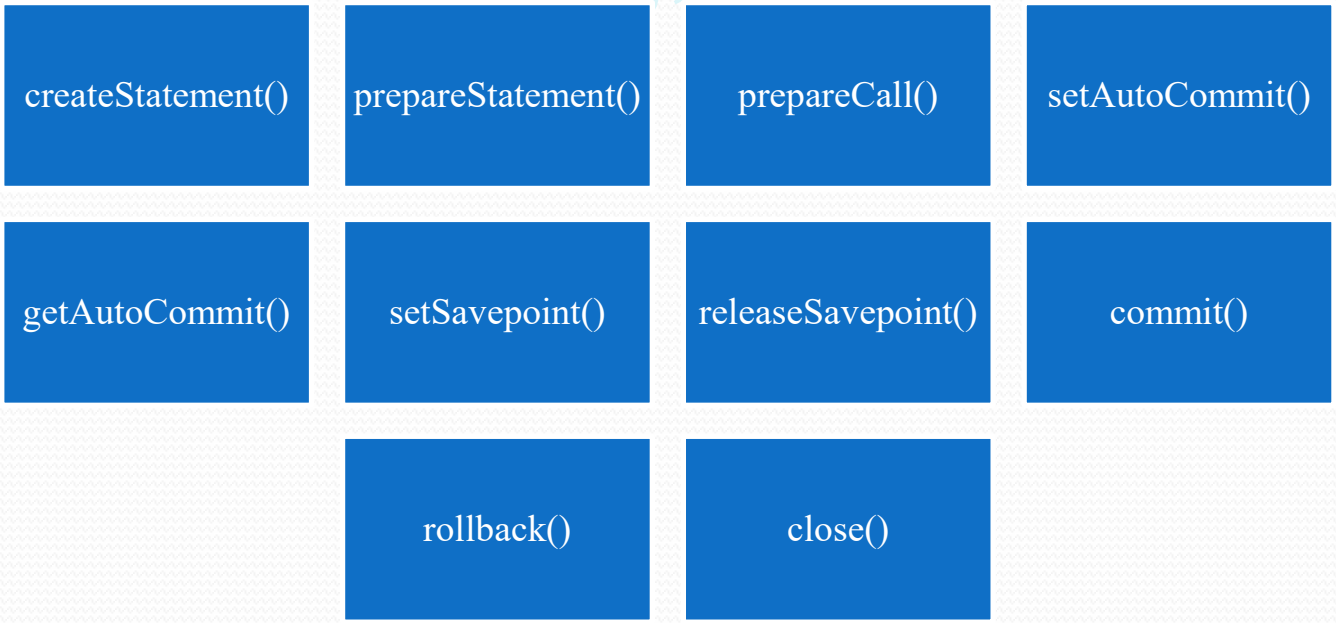
C:\oraclexe\app\oracle\product\11.2.0\server\network\ADMIN

Steps for establishing Database Connection

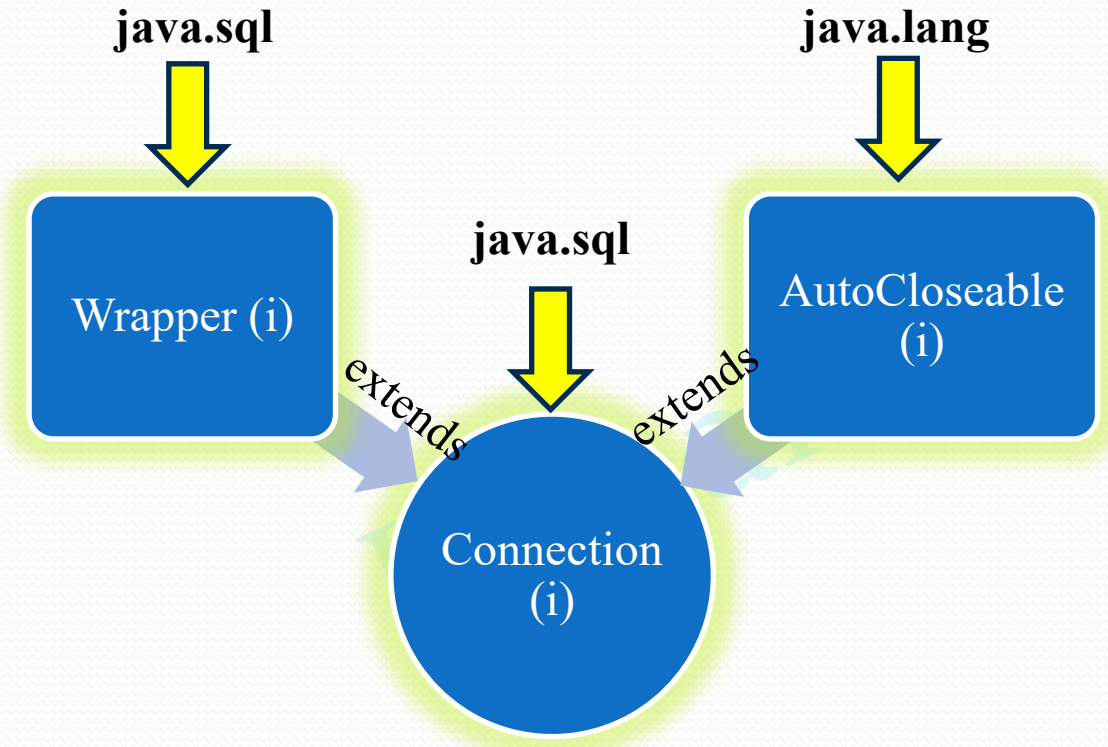
Step 1	Loading the driver **Class.*forName*("oracle.jdbc.OracleDriver");**
Step 2	Creating Connection
Step 3	Preparing Statement
Step 4	Executing query
Step 5	Closing the Connection

Understanding Connection Interface

- **java.sql** package provides the JDBC APIs for accessing and processing data which is stored in the database.
- **java.sql.Connection** interface is a crucial part of the JDBC API.
- It is used to create a connection to a database, by which we can communicate from java program to databases.
- The following are list of important methods in **java.sql.Connection.**



Hierarchy of Connection Interface



- The Connection interface is used to establish a session with a database.
- It acts as a bridge between our Java application and the database

- we use **getConnection()** from **'java.sql.DriverManager'** class to get implementation Object for **Connection** Interface.

Method Signature:

public static Connection getConnection() (String url,String user, String password) throws SQLException

Method Syntax:

Connection con=DriverManager.getConnection("DBurl","Uname","Upwd");

- **DBurl** : jdbc:oracle:thin:@localhost:1521:xe / orcl
- **Uname** : system
- **Upwd** : manager

JDBC statements

- JDBC statements are objects in Java that allows us to execute SQL queries and commands to interact with database.
- They act as a bridge between our Java application and the underlying database system.
- There are ‘3’ types of statements are present

  - Statement
  - PreparedStatement
  - CallableStatement

Understanding Statement Interface

- The **Statement** interface is a part of the JDBC API which is used to execute simple SQL queries (Create, Insert, Select, Update, Delete) **that don't need to be executed repeatedly**.
- A Statement object is created using the **createStatement()** method of the **Connection** interface.
- As createStatement() is a non-static method we need to access that method with Connection reference.


```
Connection con=DriverManager.getConnection("DBurl","Uname","Upwd");
Statement stm=con.createStatement();
```

- There are mainly ‘3’ important methods are present in Statement Interface

- ***executeQuery(String sql):*** Which is used to execute selection group SQL queries. It returns the values in a ResultSet Object

```
ResultSet rs=stm.executeQuery(“selection query”);
```

- ***executeUpdate(String sql):*** Which is used to execute NON-Selection group SQL queries. It returns int representing the number of rows affected by the execution of the statement

```
int rowCount=stm.executeUpdate(“non-Selection query”);
```

- ***execute(String sql):*** Which is used to execute SQL queries that could return different types of results. (Dynamic SQL Execution)

```
boolean flag=stm.execute(“Dynamic SQL query”);
```

Understanding PreparedStatement Interface

- The **PreparedStatement** interface is a part of the JDBC API, which is used for executing parameterized SQL queries.
- We need to use PreparedStatement if we want to execute the same SQL query multiple times with different input parameters.
- PreparedStatement object is created by using the **prepareStatement()** method of the **Connection** interface.
- As prepareStatement() is a non-static method we need to access that method with Connection reference.

```
Connection con=DriverManager.getConnection("DBurl","Uname","Upwd");
PreparedStatement pstm=con.prepareStatement("SQL QUERY FORMAT");
```

- There are mainly ‘2’ important methods are present in PreparedStatement Interface

- ***executeQuery():*** Which is used to execute selection group SQL queries. It returns the values in a ResultSet Object

```
ResultSet rs= pstm.executeQuery();
```

- ***executeUpdate():*** Which is used to execute NON-Selection group SQL queries. It returns int representing the number of rows affected by the execution of the statement

```
int rowCount= pstm.executeUpdate();
```

ResultSet Interface

Understanding ResultSet Interface

- ResultSet is an interface provided by the java.sql package.
- It represents the results obtained from the database query
- The ResultSet object is used to retrieve and manipulate the data returned by a SELECT query in a relational database

Instantiation Using Statement:

```
ResultSet rs = stm.executeQuery("SELECT * FROM Employee");
```

Instantiation Using PreparedStatement:

```
ResultSet rs = ps.executeQuery();
```

Types of ResultSet Objects in JDBC

- Based on Control and Navigation on ResultSet Objects, they can be categorized in to two types

    1) Non-Scrollable ResultSet (Only Forward direction)

    2) Scrollable ResultSet (Both the directions & jumps to specific row)

Non-Scrollable ResultSet:

- A Non-Scrollable ResultSet allows you to move only in a forward direction through the data. We cannot go back to previous rows or jump to a specific row.
- It is useful when we only need to read the data sequentially from start to finish, without any need to revisit previous rows

Scrollable ResultSet:

- A Scrollable ResultSet allows to navigate both forward and backward through the rows, jump to a specific row, or move to the first or last row.
- We need to use this if we want access the rows in a non-sequential order.
- We use the following syntaxes to generate Scrollable ResultSet Objects

Syntax using Statement:

```
Statement stm = con.createStatement(type,mode);
```

Syntax using PreparedStatement:

```
PreparedStatement ps = con.prepareStatement("query", type, mode);
```

- **‘type’ parameter:**

➢ It specifies the ***movement*** and ***behavior*** of the cursor within the ResultSet object, indicating how we can navigate through the rows of data.

➢ There are ‘3’ fields are there for **type** parameter

1. public static final int **TYPE_FORWARD_ONLY**;------1003
2. public static final int **TYPE_SCROLL_INSENSITIVE**;----1004
3. public static final int **TYPE_SCROLL_SENSITIVE**;------1005

- **‘mode’ parameter:**

➢ It specifies the action to be performed on ResultSet Object

➢ There are ‘2’ fields are there for **mode** parameter

1. public static final int **CONCUR_READ_ONLY**;----1007
2. public static final int **CONCUR_UPDATABLE**;----1008

Important methods in Scrollable ResultSet

Method	Description	Use Case
afterLast()	Moves the cursor to a position after the last row in the ResultSet	Useful for iterating backward through the ResultSet
beforeFirst()	Moves the cursor to a position before the first row in the ResultSet	Useful for re-iterating from the beginning of the ResultSet
last()	Moves the cursor to the last row in the ResultSet	Quickly access the last row, such as the most recent entry.
first()	Moves the cursor to the first row in the ResultSet	Useful for accessing the first record directly
previous()	Moves the cursor to the previous row in the ResultSet	Iterating backward through the rows
next()	Moves the cursor to the next row in the ResultSet	Commonly used to iterate forward through the rows.
absolute(int row)	Moves the cursor to the specified row number. Positive numbers count from the start, negative from the end.	Jump directly to a specific row by its position.
relative(int rows)	Moves the cursor by a relative number of rows from the current position.	Useful for moving a specific number of rows forward or backward