Java Frameworks Package

1. HIBERNATE ,JPA,JPA2

2. SPRING

3. SPRING BOOT2

4. MICRO SERVICES

5. WEB SERVICES JAX-RS,JAXB

4. SERVLETS

5. JSP

ANGULER8,9

REACT JS

CORE JAVA , JDBC,DGP

Syllabus: J2SE: [Core Java]:

1. Introduction:

1. Java History
2. Differences between java and others
3. Java Features
4. Java Naming Conventions
5. Java Programming Format

2. First Java Application Development:

1. Java Installation
2. Editor
3. Java Application and Java File Saving.
4. Compile Java File
5. Execute Java Applications.

3. Language Fundamentals:

1. Tokens
2. Identifiers
3. Literals
4. Key Words /Reserved Words

1. Member Inner class
2. Static Inner class
3. Method local Inner class
4. Anonymous Inner class

6. Wrapper classes:

1. Byte, Short, Integer, Long, Float, Double, Boolean, Character

7. Packages:

1. What is a package?
2. Adv. of packages
3. Types of packages
4. Jar files preparation
5. Executable Jar files
6. Batch files preparation

8. String manipulations:

1. String
2. String Buffer
3. String Builder
4. String to kenizer

9. Exception Handling:

1. Error VS Exception
2. Exception Def.
3. Types of Exceptions
4. Checked Exception VS Unchecked Exception
5. Throw Vs throws
6. try-catch-finally
7. Custom Exceptions
8. Java7 Features in Exception Handling

10. Multi-Threading:

1. Process Vs Processor Vs Procedure
2. Single Processing Mech. Vs Multi Processing Mech.
3. Single Thread model And Multi Thread Model
4. Thread Design
5. Thread lifecycle
6. Thread class library
7. Daemon Thread
8. Synchronization
9. Inter Thread communication
10. Deadlocks

11. IOStreams:

1. What is stream?
2. Types of Streams?
3. File Input Stream Vs File Output Stream
4. File Reader Vs File Writer
5. File Vs Random Access File
6. Serialization vs Deserialization
7. Externalization

13. Collection Framework:

1. Collection Arch.
2. List and its implementations
3. Set and its implementations
4. Map and its implementations
5. Queue and its implementations
6. Iterators

14. AWT:

1. Text Field, Text Area, Button, Label, Check Box, List.

15. Swing:

1. J Text Field, J Password Field, J Check Box, J Radio Button, J Color Chooser.
2. Event Delegation Model

16. I18N:

1. Number Format
2. Date Format
3. Resource Bundle

17. Reflection API:

1. Class
2. Field
3. Method
4. Constructor

18. Annotations:

1. What is Annotation?
2. Adv of annotations
3. Comments Vs Annotations
4. Types of annotations

19. Remote Method Invocation[RMI]:

1. Introduction
2. RMI Architecture
3. Steps to Design RMI Application
4. Parameters in Remote methods

20. Regular Expressions:

1. Introduction
2. Pattern
3. Character
4. Quantifiers

21. Garbage Collection:

1. Introduction
2. Approaches to make an object for GC
3. Methods for requesting JVM to run GC
4. Finalization

22. JVM Arch.

1. Class Loading Sub System
2. Memory Management System
3. Execution Engine
4. Java Native Interface
5. Java Native library

23. Generics:

1. Introduction
2. Generic Classes
3. Generic Methods & Wild Card Character.
4. Inter Communication with Non-Generic Code

24. Java 8 Features:

1. Lambda Expressions
2. Functional Interfaces
3. Default Methods in Interfaces
4. Static Methods in Interfaces
5. Predicate
6. Function
7. Consumer
8. Method Reference & Constructor reference Double Colon (::. operator.)

9. Stream API
10. Date & Time API (Joda API.)

25. JAVA 9 NEW FEATURES:

1. The Java Shell (RPEL.)
2. The Java Platform Module System(JPMS.)
3. JLINK(JAVA LINKER.)
4. Process API Updates
5. Private Method in Interfaces.
6. Factory Methods for Collections.
7. Enhancements to Java 8 Stream API
8. Try With Resources Enhancements.
9. Diamond Operator
10. SafeVargs Annotation
11. HTTP/2 Client
12. G1 Garbage Collector

26. Basics of JDBC:

1. Introduction.
2. JDBC Drivers.
3. Steps to prepare JDBC Applications
4. JDBC Applications for CRUD Operations

Java 10 Updations:

1. Local-Variable Type Inference
2. Consolidate the JDK Forest into a Single Repository
3. Garbage-Collector Interface
4. Parallel Full GC for G1
5. Application Class-Data Sharing
6. Thread-Local Handshakes
7. Remove the Native-Header Generation Tool (javah)
8. Additional Unicode Language-Tag Extensions
9. Heap Allocation on Alternative Memory Devices
10.Experimental Java-Based JIT Compiler
11.Root Certificates
12.Time-Based Release Versioning

Java 11 Version Updations:

1. Running Java File with single command
2. New utility methods in String class
3. Local-Variable Syntax for Lambda Parameters
4. Nested Based Access Control
5. HTTP Client

6. Reading/Writing Strings to and from the Files
7. Flight Recorder

Java 12 Features:

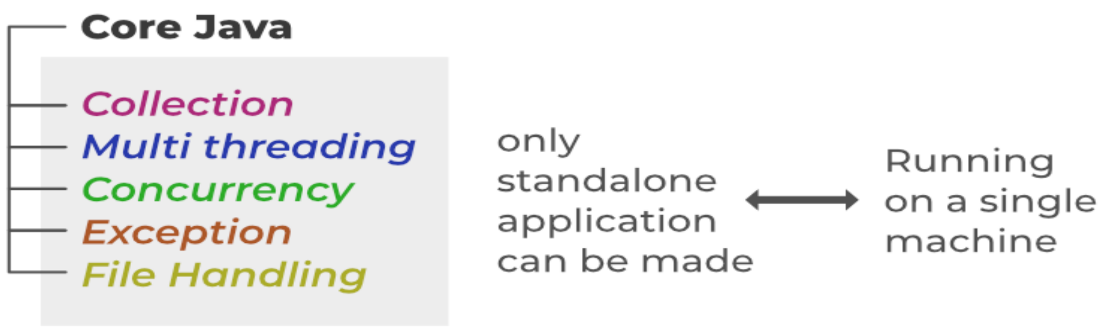
1. Switch Expressions
2. File mismatch() Method
3. Compact Number Formatting
4. Teeing Collectors in Stream API
5. Java Strings New Methods – indent(), transform(), describeConstable(), and resolveConstantDesc().
6. JVM Constants API
7. Pattern Matching for instanceof
8. Raw String Literals is Removed From JDK 12.

Java 13 Updations:

1. Text Blocks
2. New Methods in String Class for Text Blocks
3. Switch Expressions Enhancements
4. Reimplement the Legacy Socket API
5. Dynamic CDS Archive
6. ZGC: Uncommit Unused Memory
7. FileSystems.newFileSystem() Method

ID's

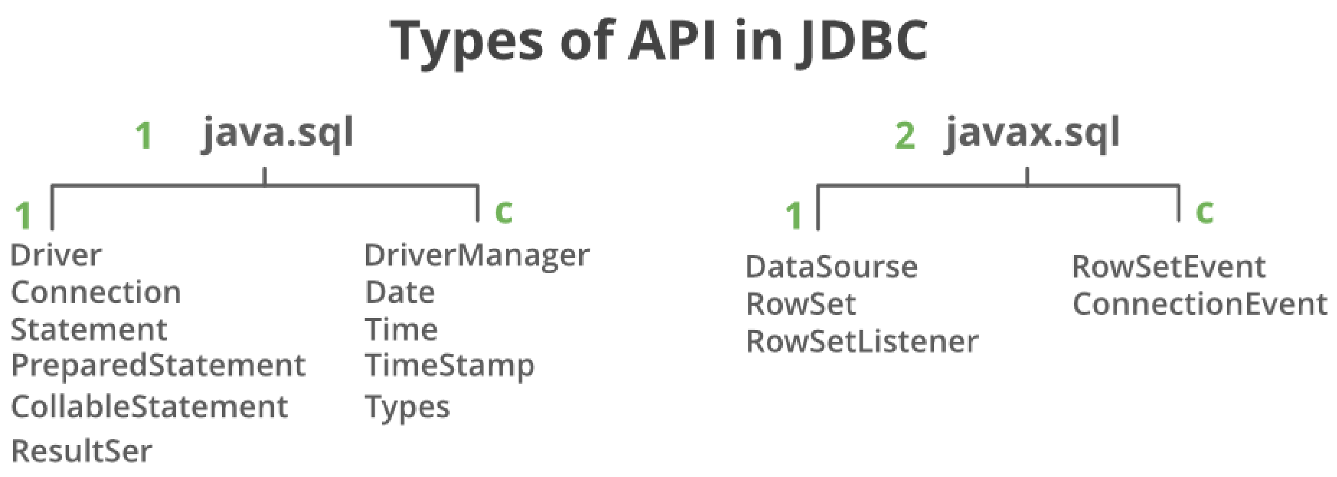
1. **Eclipse**
2. **VSC**
3. **EDITPLUS**

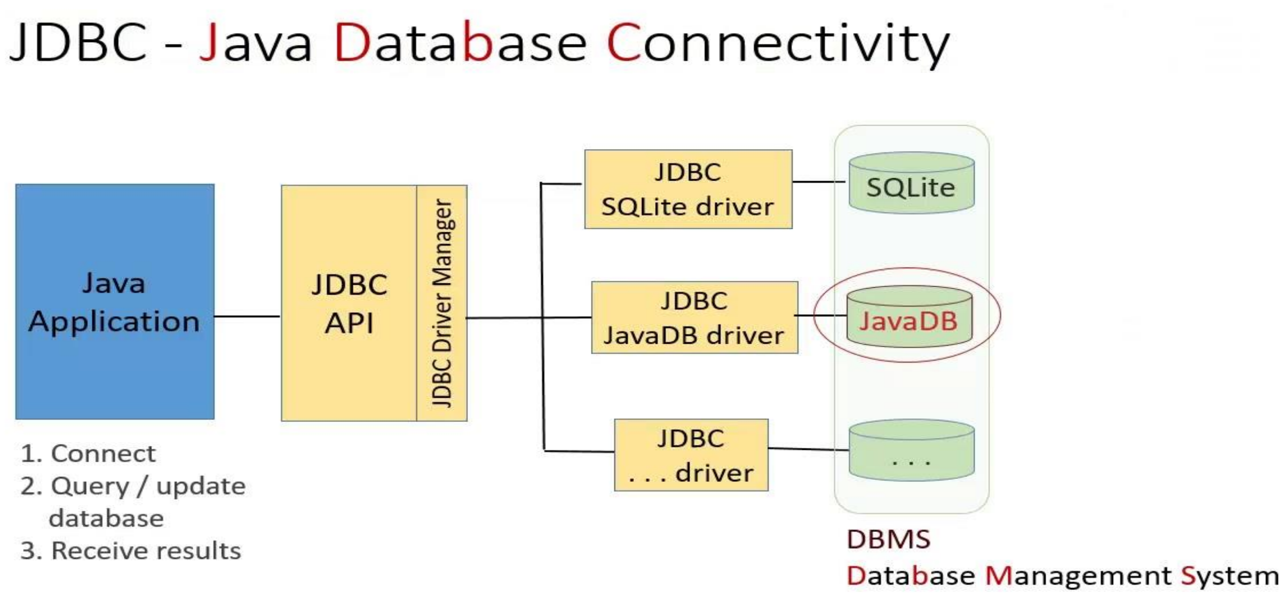


JDBC API 4.0 mainly provides 2 important packages:

- **java.sql**
- **javax.sql**

- What is JDBC
- Need of JDBC
- Datatypes in JDBC
- JDBC Architecture
- JDBC Environment Setup
- Steps to connect JDBC (DONE)
  JDBC Connection Steps
    - #1) Import Packages
    - #2) Load Driver
      - (i) Class.forName()
      - (ii) DriverManager.registerDriver()
    - #3) Establish Connection
    - #4) Create And Execute StatementS
      - (i) Create Statement
      - (ii) Execute The Query
    - #5) Retrieve Results
    - #6) Close Connection
  - Java JDBC Connection Example
    - Create Table
    - Insert Data Into Table
    - Java Program
    - Frequently Asked Questions
- JDBC example as implementation (DONE)





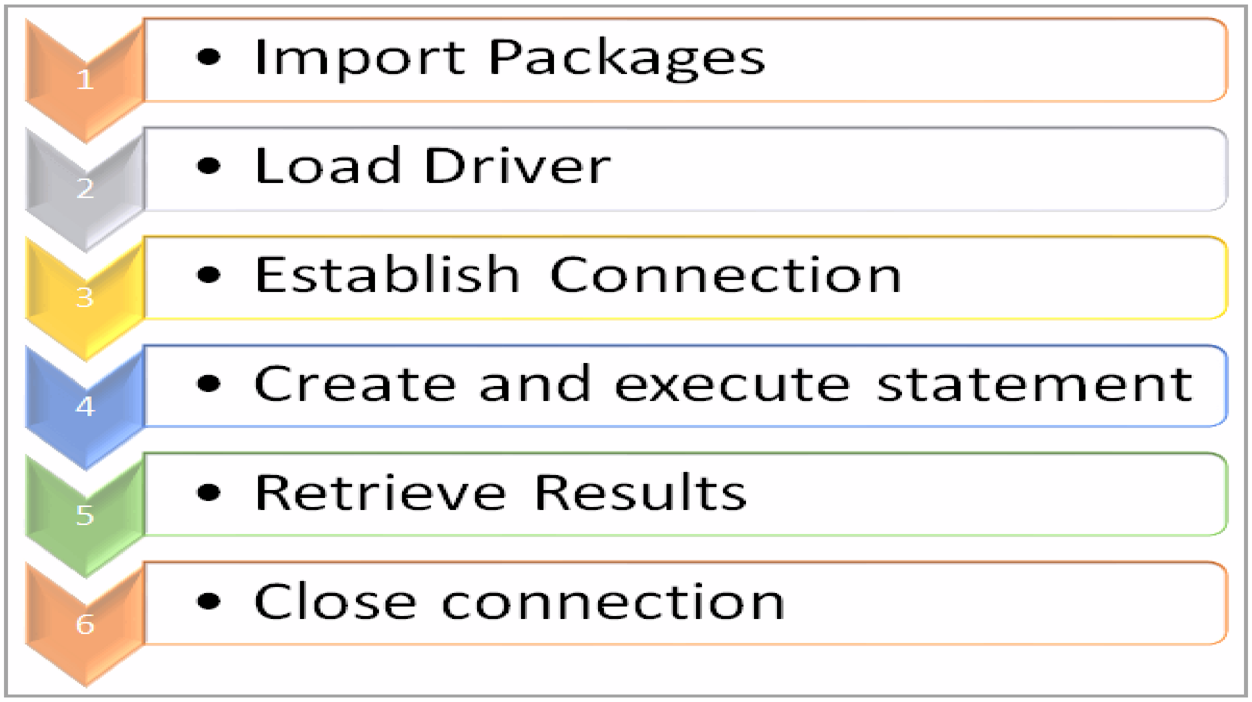
Key Classes and Interfaces

- JDBC Connection Interface
- JDBC Statement Interface
- JDBC PreparedStatement Interface
- JDBC CallableStatement Interface
- JDBC ResultSet Interface with Examples
- JDBC ResultSetMetaData Interface
- JDBC DatabaseMetaData Interface

- JDBC DriverManager Class

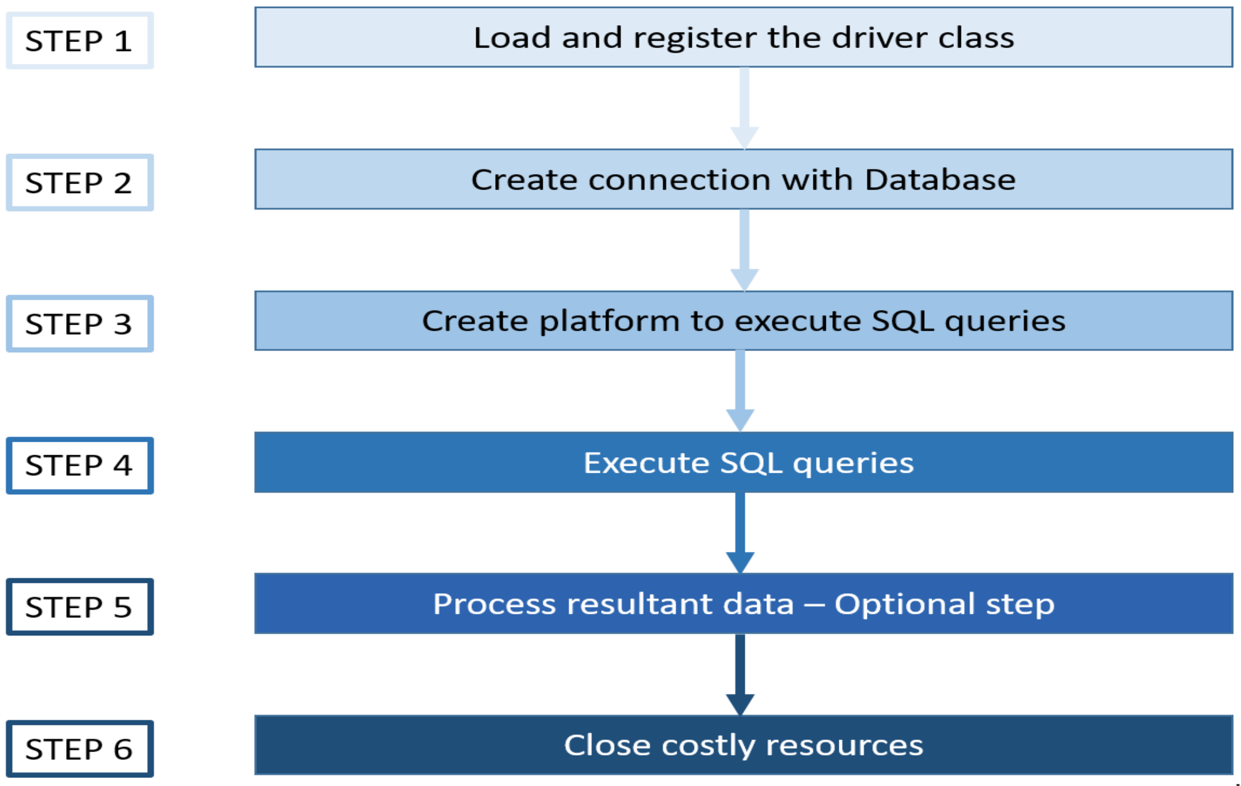
There are 4 different types of JDBC drivers:

1. **Type 1: JDBC-ODBC bridge driver**
2. **Type 2: Java + Native code driver**
3. **Type 3: All Java + Middleware translation driver**
4. **Type 4: All Java driver.**



1. **Load the JDBC driver.**
2. **Connection.**
3. **Statement.**
4. **Execute statement.**
5. **Close database connection.**

JDBC connection steps in java:

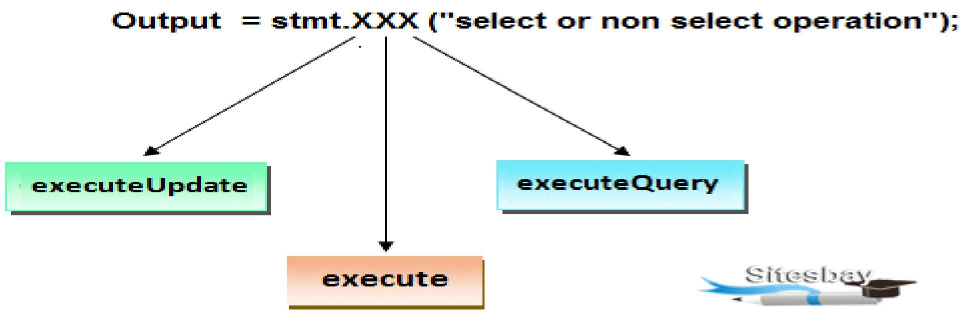

STEP 1
Load and register the driver class
STEP 2
Create connection with Database
STEP 3
Create platform to execute SQL queries
STEP 4
Execute SQL queries
STEP 5
Process resultant data – Optional step
STEP 6
Close costly resources

There are 5 steps to connect any java application with the database using JDBC

1. **Load the JDBC driver class or register the JDBC driver.**
2. **Establish the connection**
3. **Create a statement**
4. **Execute the sql commands on database and get the result**
5. **Close the connection**

Executing queries

- **executeUpdate():** Used for non-select operations.
- **executequery():** Used for select operation.
- **execute():** Used for both select or non-select operation.



Methods of Statement interface:

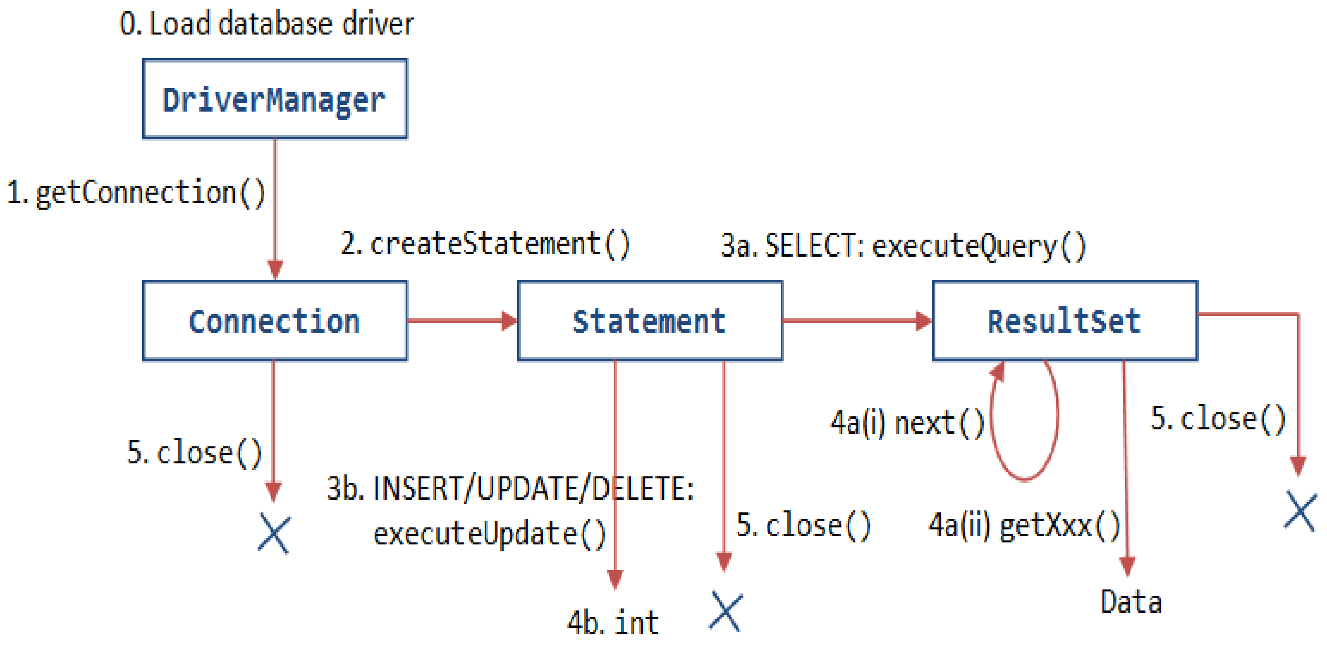
1. execute(String SQL): It is used to execute SQL DDL statements.

Syntax:
public boolean execute(String SQL)

2. executeQuery(String SQL): It is used to execute the select query and returns an object of ResultSet.
Syntax:
public ResultSet executeQuery(String SQL)
3. executeUpdate(String SQL): It is used to execute the query like inset, update, delete etc. It returns the no. of affected rows.
Syntax:
public int executeUpdate(String SQL)
4. executeBatch(): It is used to execute the batch of commands.
Syntax:
public int[] executeBatch()

Technologies used

1. **JDK - 1.8,9,11 or later**
2. **MySQL - 5.7.12, ORACLE11G JAR-(MySQL connecter, ojdbc6, ojdbc8, ojdbc14.jar)**
3. **IDE - Eclipse Neon**
4. **JDBC API - 4.2**



Steps to connect database in java using JDBC:

1. **Load and Register the JDBC driver.**
2. **Connection.**
3. **Create Statement object**
4. Execute the SQL commands DDL DRL DML
5. **Close database connection.**

Class.forName("oracle.jdbc.driver.OracleDriver");

DB Name	JDBC Driver Name
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver

1. Load the JDBC driver:

First step is to load or register the JDBC driver for the database. Class class provides forName() method to dynamically load the driver class.
Syntax:
Class.forName("oracle.jdbc.driver.OracleDriver");

2. Create connection:

Second step is to open a database connection. DriverManager class provides the facility to create a connection between a database and the appropriate driver.To open a database connection we can call getConnection() method of DriverManager class.
Syntax:
Connection connection = DriverManager.getConnection(url, user, password)

To create a connection with Oracle database:
Syntax:
Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","user","password");

3. Create statement:

The statement object is used to execute the query against the database. Connection interface acts as a factory for statement object., *CallableStatement,* and *PreparedStatement types* .
commit() and rollback() etc.
Syntax:
Statement stmt=conn.createStatement();

4. Execute statement:

Statement interface provides the methods to execute a statement.

To execute a statement for select query use below:
Syntax:
ResultSet resultSet = stmt.executeQuery(selectQuery);

5. Close database connection:

After done with the database connection we have to close it. Use close() method of Connection interface to close database connection. be closed automatically when we close the connection object.
Syntax:
connection.close();

```
import java.sql.*;
class CreateTable
{
public static void main(String[] args) throws Exception
{
//step-1

Class.forName("oracle.jdbc.driver.OracleDriver ");
System.out.println("driver is laoded");

//step-2
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","user","password");
System.out.println("connection is established");

//step-3
Statement stmt=con.createStatement();
System.out.println("statement object is cretaed");

//step-4
int i=stmt.executeUpdate("create table student(sid number(3),sname varchar2(10),marks number(5))");
//step-5

System.out.println("Result is="+i);
System.out.println("table is created");

//step-6
stmt.close();
con .close();
}
}
```

```
public class JDBCOracleTest {
          //JDBC and database properties.
          private static final String DB_DRIVER =
                              "oracle.jdbc.driver.OracleDriver";
          private static final String DB_URL =
                            "jdbc:oracle:thin:@localhost:1521:XE";
          private static final String DB_USERNAME = "system";
          private static final String DB_PASSWORD = "oracle";

          public static void main(String args[]){
                    Connection conn = null;
                    try{
                              //Register the JDBC driver
                              Class.forName(DB_DRIVER);

                              //Open the connection
                              conn = DriverManager.
                              getConnection(DB_URL, DB_USERNAME, DB_PASSWORD);

                              if(conn != null){
                                System.out.println("Successfully connected.");
                              }else{
                                System.out.println("Failed to connect.");
                              }
                    }catch(Exception e){
                              e.printStackTrace();
                    }
          }
}
```

```
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
    DatabaseMetaData dbmd=con.getMetaData();
    System.out.println("Driver Name: "+dbmd.getDriverName());
    System.out.println("Driver Version: "+dbmd.getDriverVersion());
    System.out.println("UserName: "+dbmd.getUserName());
    System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
    System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());
    con.close();
    }catch(Exception e){ System.out.println(e);}
}
    }
```

```
String query = "create table EMPLOYEE("
                              + "EMPLOYEE_ID NUMBER(5) NOT NULL, "
                              + "NAME VARCHAR(20) NOT NULL, "
                              + "SALARY NUMBER(10) NOT NULL, "
```

```
                                + "PRIMARY KEY (EMPLOYEE_ID) )";
//execute query
                                statement.execute(query);

                                //close connection
                                statement.close();
                                conn.close();

                                System.out.println("Table created successfully.");
                      }
String query = "insert into EMPLOYEE " +
                      "(EMPLOYEE_ID, NAME, SALARY) " +
                      "values (1, 'Harish Yadav', 50000)";

//execute query
                                statement.executeUpdate(query);

                                //close connection
                                statement.close();
                                conn.close();

                      System.out.println("Record inserted successfully.");

String query = "update EMPLOYEE set " +
                      "NAME = 'Abhishek' " +
                      "where EMPLOYEE_ID = 1 ";
//execute query
                                statement.executeUpdate(query);

                                //close connection
                                statement.close();
                                conn.close();

                           System.out.println("Record updated successfully.");

String query = "select EMPLOYEE_ID, NAME from EMPLOYEE";

//execute query
                                ResultSet rs = statement.executeQuery(query);
                                while (rs.next()) {
                                  String empId = rs.getString("EMPLOYEE_ID");
                                  String empName = rs.getString("NAME");

                                  System.out.println("EmpId : " + empId);
                                  System.out.println("EmpName : " + empName);
                                }

                                //close connection
                                statement.close();
                                conn.close();

String query = "delete EMPLOYEE " +
                      "where EMPLOYEE_ID = 1 ";

//execute query
```

```
                                        statement.executeUpdate(query);

                                        //close connection
                                        statement.close();
                                        conn.close();

                                System.out.println("Record deleted successfully.");


String query1 = "insert into EMPLOYEE " +
                                                    "(EMPLOYEE_ID, NAME, SALARY) " +
                                                    "values (1, 'Harish Yadav', 50000)";

                            String query2 = "insert into EMPLOYEE " +
                                                    "(EMPLOYEE_ID, NAME, SALARY) " +
                                                    "values (5, 'Abhishek Rathor', 50000)";

//set auto commit to false
                                        conn.setAutoCommit(false);

                                        //add queries to batch
                                        statement.addBatch(query1);
                                        statement.addBatch(query2);

                                        //execute batch
                                        statement.executeBatch();

                                        //commit
                                        conn.commit();

                                        //close connection
                                        statement.close();
                                        conn.close();

                                System.out.println("Records inserted successfully.");
```

Advantages of PreparedStatement:

1. **Parameterized query: Provides the facility of parameterized query.**
2. **Reusable: PreparedStatement can be easily used with new parameters.**
3. **Performance: It increases the performance because of database statement caching.**

Difference between Statement and PreparedStatement in jdbc:

Statement	**PreparedStatement**
1. Statement not executes the parameterized query. 2. Relational DB uses following 4 step to execute a query: a. Parse the query. b. Compile the query. c. Optimize/Plan the query.	1. PreparedStatement can execute the parameterized query. 2. Relational DB uses following 4 step to execute a query: a. Parse the query. b. Compile the query. c. Optimize/Plan the query.

d. Execute the query. **A statement always executes the all four steps.** 3. No database statement caching in case of statement.	d. Execute the query. **PreparedStatement pre-executes first three steps in the execution.** 3. It provides the database statement caching the execution plans of previously executed statements. Hence database engine can reuse the plans for statements that have been executed previously.

```
try{
                //get connection
                conn = JDBCUtil.getConnection();

                //create preparedStatement
                preparedStatement = conn.prepareStatement(query);

                //execute query
                preparedStatement.execute();

                //close connection
                preparedStatement.close();
                conn.close();

            System.out.println("Table created successfully.");

String query = "insert into EMPLOYEE " +
                "(EMPLOYEE_ID, NAME, SALARY) " +
                "values (?,?,?)";

try{
                //get connection
                conn = JDBCUtil.getConnection();

                //create preparedStatement
                preparedStatement = conn.prepareStatement(query);

                //set values
                preparedStatement.setInt(1, 1);
                preparedStatement.setString(2, "Bharat");
                preparedStatement.setInt(3, 62000);

                //execute query
                preparedStatement.executeUpdate();

                //close connection
                preparedStatement.close();
                conn.close();

            System.out.println("Record inserted successfully.");
```

```
String query = "update EMPLOYEE set " +
                    "SALARY = ? " +
                    "where EMPLOYEE_ID = ? ";

                    try{
                              //get connection
                              conn = JDBCUtil.getConnection();

                              //create preparedStatement
                              preparedStatement = conn.prepareStatement(query);

                              //set values
                              preparedStatement.setInt(2, 1);
                              preparedStatement.setInt(1, 65000);

                              //execute query
                              preparedStatement.executeUpdate();

                              //close connection
                              preparedStatement.close();
                              conn.close();

                        System.out.println("Record updated successfully.");

String query = "delete EMPLOYEE " +
                    "where EMPLOYEE_ID = 1 ";

                    try{
                              //get connection
                              conn = JDBCUtil.getConnection();

                              //create preparedStatement
                              preparedStatement = conn.prepareStatement(query);

                              //execute query
                              preparedStatement.executeUpdate();

                              //close connection
                              preparedStatement.close();
                              conn.close();

                       System.out.println("Record deleted successfully.");



String query = "insert into EMPLOYEE " +
                              "(EMPLOYEE_ID, NAME, SALARY) " +
                              "values (?,?,?)";

                    try{
                              //get connection
                              conn = JDBCUtil.getConnection();
```

```
				//set auto commit to false
				conn.setAutoCommit(false);

				//create preparedStatement
				preparedStatement = conn.prepareStatement(query);

				//set values
				preparedStatement.setInt(1, 1);
				preparedStatement.setString(2, "Bharat");
				preparedStatement.setInt(3, 62000);
				preparedStatement.addBatch();

				//set values
				preparedStatement.setInt(1, 4);
				preparedStatement.setString(2, "Bharti");
				preparedStatement.setInt(3, 52000);
				preparedStatement.addBatch();

				//execute query
				preparedStatement.executeBatch();

				//commit
				conn.commit();

				//close connection
				preparedStatement.close();
				conn.close();

			System.out.println("Record inserted successfully.");


String query = "select * from FILESTORE " +
			"where FILE_ID = 2";

			try{
				//get connection
				conn = JDBCUtil.getConnection();

				//create preparedStatement
				preparedStatement =
					conn.prepareStatement(query);

				//execute query
				ResultSet resultSet =
					preparedStatement.executeQuery();

				resultSet.next();

				Clob clob = resultSet.getClob(2);
				Reader reader = clob.getCharacterStream();

				FileWriter fileWriter =
					new FileWriter("F:\\savedFile.txt");
```

```
                                int i;
                                while((i=reader.read())!=-1){
                                          fileWriter.write((char)i);
                                }

                          System.out.println("File retrieved successfully.");

                                //close connection
                                fileWriter.close();
                                preparedStatement.close();
                                conn.close();
String createTableQuery = "create table FILESTORE("
                                + "FILE_ID NUMBER(5) NOT NULL, "
                                + "NAME CLOB NOT NULL, "
                                + "PRIMARY KEY (FILE_ID) )";

                      try{
                                //get connection
                                conn = JDBCUtil.getConnection();

                                //create preparedStatement
                                preparedStatement =
                                          conn.prepareStatement(createTableQuery);

                                //execute query for create table
                                preparedStatement.execute();
                          System.out.println("Table created successfully.");

                                String storeFileQuery = "insert into FILESTORE "
                                          + "values (?,?)";
                                preparedStatement =
                                          conn.prepareStatement(storeFileQuery);

                                //Read source file
                                File file = new File("F:\\test.txt");
                                FileReader fileReader = new FileReader(file);

                                preparedStatement.setInt(1,2);
                                preparedStatement.setCharacterStream(2,
                                               fileReader,(int)file.length());

                                preparedStatement.executeUpdate();
                          System.out.println("File stored successfully.");

                                //close connection
                                preparedStatement.close();
                                conn.close();
```

Explain JDBC core components.

The JDBC API consists of the following core components:
1. JDBC Drivers
2. Connections
3. Statements
4. ResultSets
See more at: JDBC Components.

What are different types of JDBC Drivers?

JDBC driver:
A driver is a software component that provides the facility to interact java application with the database.
Types of JDBC drivers:
1. JDBC-ODBC bridge driver.
2. Native-API driver.
3. Network-Protocol driver.
4. Thin driver.
See more at: JDBC Driver.

What are the main steps in java to make JDBC connectivity?

Steps to connect database in java using JDBC are given below:
1. Load the JDBC driver.
2. connection.
3. statement.
4. Execute statement.
5. Close database connection.

See more at: Steps to connect database in java.

What is JDBC Statement?

The JDBC statement is used to execute queries against the database. Statement is an interface which provides the methods to execute queries. We can get a statement object by invoking the createStatement() method of Connection interface.
Syntax: Statement stmt=conn.createStatement();
See more at: JDBC Statement interface.

What is the difference between execute, executequery, executeupdate?

executeQuery()—for getting the data from database.
executeUpdate()—for insert,update,delete.
execute()—any kind of operations.
boolean execute(): Executes the SQL statement in this Prepared Statement object, which may be any kind of SQL statement.
ResultSet executeQuery(): Executes the SQL query in this Prepared Statement object and returns the ResultSet object generated by the query.
int executeUpdate(): Executes the SQL statement in this Prepared Statement object, which must be an SQL INSERT, UPDATE or DELETE statement; or an SQL statement that returns nothing, such as a DDL statement.

What is JDBC PreparedStatement?

The JDBC PreparedStatement is used to execute parameterized queries against the database. PreparedStatement is an interface which provides the methods to execute parameterized queries. A parameter is represented by ? symbol in JDBC. PreparedStatement extends the Statement interface. We can get a PreparedStatement object by invoking the prepareStatement() method of Connection interface.
Syntax: PreparedStatement pstmt = conn.prepareStatement(SQL);

Explain JDBC Savepoint.

A savepoint represents a point that the current transaction can roll back to. Instead of rolling all of its changes back, it can choose to roll back only some of them. For example, suppose you
• start a transaction.

• insert 20 rows into a table.
• set a savepoint.
• insert another 10 rows.
• rollback to the savepoint.
• commit the transaction.
After doing this, the table will contain the first 20 rows you inserted. The other 10 rows will have been deleted by the rollback. A savepoint is just a marker that the current transaction can roll back to.

What is the use of blob and clob datatypes in JDBC?

The blob and clob datatypes in JDBC are used to store large amount of data into database like images, movie etc which are extremely large in size.

What is Connection Pooling?

Connection Pooling is a technique used for reuse of physical connections and reduced overhead for your application. Connection pooling functionality minimizes expensive operations in the creation and closing of sessions. Database vendor's help multiple clients to share a cached set of connection objects that provides access to a database. Clients need not create a new connection every time to interact with the database.

How do you implement connection pooling?

If you use an application server like WebLogic, WebSphere, jBoss, Tomcat. , then your application server provides the facilities to configure for connection pooling. If you are not using an application server then components like Apache Commons DBCP Component can be used.

What is 2 phase commit?

When we work in distributed systems where multiple databases are involved, we are required to use 2 phase commit protocol. 2 phase commit protocol is an atomic commitment protocol for distributed systems.

In the first phase, transaction manager sends commit-request to all the transaction resources. If all the transaction resources are OK, then transaction manager commits the transaction changes for all the resources. If any of the transaction resource responds as Abort, then the transaction manager can rollback all the transaction changes.

What are the different types of locking in JDBC?

Optimistic Locking: Optimistic locking lock the record only when update take place. Optimistic locking does not use exclusive locks when reading.
Pessimistic locking: Record are locked as it selects the row to update.

What is a "dirty read"?

In typical database transactions, say one transaction reads and changes the value while the second transaction reads the value before committing or rolling back by the first transaction. This reading process is called as 'dirty read'.

What are the JDBC API components?

The java.sql package contains following interfaces and classes for JDBC API.

Interfaces:

- **Connection:** The Connection object is created by using getConnection() method of DriverManager class DriverManager is the factory for connection.
- **Statement:** The Statement object is created by using createStatement() method of Connection class. The Connection interface is the factory for Statement.
- **PreparedStatement:** The PrepareStatement object is created by using prepareStatement() method o Connection class. It is used to execute the parameterized query.
- **ResultSet:** The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points before the first row. The executeQuery() method of Statement interface returns the ResultSet object.
- **ResultSetMetaData:** The object of ResultSetMetaData interface cotains the information about the data (table) such as numer of columns, column name, column type, etc. The getMetaData() method of ResultSe returns the object of ResultSetMetaData.
- **DatabaseMetaData:** DatabaseMetaData interface provides methods to get metadata of a database such as the database product name, database product version, driver name, name of the total number of tables, the name of the total number of views, etc. The getMetaData() method of Connection interface returns the objec of DatabaseMetaData.
- **CallableStatement:** CallableStatement interface is used to call the stored procedures and functions. We car have business logic on the database through the use of stored procedures and functions that will make the performance better because these are precompiled. The prepareCall() method of Connection interface returns the instance of CallableStatement.

Classes:

- **DriverManager:** The DriverManager class acts as an interface between the user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains several methods to keep the interaction between the user and drivers.
- **Blob:** Blob stands for the binary large object. It represents a collection of binary data stored as a single entity in the database management system.
- **Clob:** Clob stands for Character large object. It is a data type that is used by various database managemen systems to store character files. It is similar to Blob except for the difference that BLOB represent binary data such as images, audio and video files, etc. whereas Clob represents character stream data such as characte files, etc.
- **SQLException** It is an Exception class which provides information on database access errors.

5) What are the JDBC statements?

In JDBC, Statements are used to send SQL commands to the database and receive data from the database. There are various methods provided by JDBC statements such as execute(), executeUpdate(), executeQuery, etc. which helps you to interact with the database.

There is three type of JDBC statements given in the following table.

Statements	Explanation
Statement	Statement is the factory for resultset. It is used for general purpose access to the database. It executes a static SQL query at runtime.
PreparedStatement	The PreparedStatement is used when we need to provide input parameters to the query at runtime.
CallableStatement	CallableStatement is used when we need to access the database stored procedures. It can also accept runtime parameters.

10) What are the differences between execute, executeQuery, and executeUpdate?

execute	executeQuery	executeUpdate
The execute method can be used for any SQL statements(Select and Update both).	The executeQuery method can be used only with the select statement.	The executeUpdate method can be used to update/delete/insert operations in the database.
The execute method returns a boolean type value where true indicates that the ResultSet s returned which can later be extracted and false indicates that the integer or void value is returned.	The executeQuery() method returns a ResultSet object which contains the data retrieved by the select statement.	The executeUpdate() method returns an integer value representing the number of records affected where 0 indicates that query returns nothing.

1. What is JDBC API and when do we use it?
2. What are different types of JDBC Drivers?
3. How does JDBC API helps us in achieving loose coupling between Java Program and JDBC Drivers API?
4. What is JDBC Connection? Explain steps to get Database connection in a simple java program.
5. What is the use of JDBC DriverManager class?
6. How to get the Database server details in java program?
7. What is JDBC Statement?
8. What is the difference between execute, executeQuery, executeUpdate?
9. What is JDBC PreparedStatement?
10. How to set NULL values in JDBC PreparedStatement?
11. What is the use of getGeneratedKeys() method in Statement?
12. What are the benefits of PreparedStatement over Statement?
13. What is the limitation of PreparedStatement and how to overcome it?
14. What is JDBC ResultSet?
15. What are different types of ResultSet?
16. What is the use of setFetchSize() and setMaxRows() methods in Statement?
17. How to use JDBC API to call Stored Procedures?
18. What is JDBC Batch Processing and what are it's benefits?
19. What is JDBC Transaction Management and why do we need it?