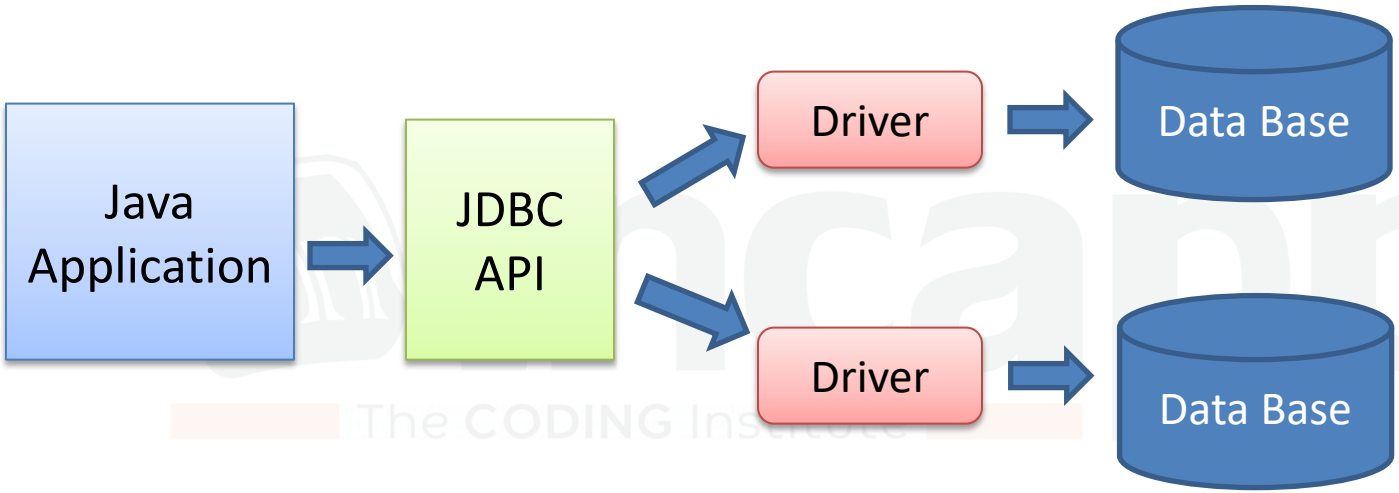
JDBC(Java Data Base Connectivity)

JDBC is an API used to connect a java program to Database (MySQL, Oracle 11g etc.).

Packages that contains JDBC API:
java.sql
javax.sql

JDBC Driver/Connector

JDBC uses driver class to take java code to database.
Just like, a car driver take us to one place to another place.



Driver is a class, that actually connect java program to data base.
JDBC is an API provided by java, that has collection of abstract methods. All methods are implemented by Driver class provided by Database vendors.
Driver is provided in jar file by Database vendors.

5 Steps for JDBC

Step-1: Load the Data Base Driver.

Class.forName("DriverName");

Step-2: Get the Data Base Connection.

Connection c=DriverManager.getConnection("Database Path", "ID", "Password");

Step-3: Create Statement to fire or execute SQL queries.

Statemet st=c.createStatement();

Step-4: Fire/Execute SQL queries.

st.executeUpdate("Insert/Create/Update/Delete SQL Query");
st.executeQuery("Read Query");

Step-5: Close Data Base connection.

c.close();

PreparedStatement vs Statement Interface

	Statement	PreparedStatement
Purpose	Executes simple SQL queries.	Executes parameterized or precompiled SQL queries.
SQL Injection	Vulnerable to SQL injection attacks.	Prevents SQL injection attacks due to parameterization.
Performance	Slower for repeated executions (query parsed each time).	Faster for repeated executions (query precompiled once).

Statement syntax:

```
Statement st=con.createStatement();
st.executeUpdate("insert into employee (eid, name, salary) values('e103', 'Rajnikant', 158000)");
```

PreparedStatement syntax:

```
PreparedStatement p=con.prepareStatement("insert into employee (eid, name, salary, age) values(?,?,?)");
p.setString(1, "e103 ");
p.setString(2,"Rajnikant");
p.setInt(3, 158000);
p.executeUpdate();
```

ResultSet Interface

The **ResultSet** interface in JDBC is used to represent the result of a database query. It acts as an iterator to navigate through rows of a result set, and provides methods to retrieve data.

Syntax:

```
ResultSet rs=st.executeQuery("select * from employee where eid= 'e101' ");
if(rs.next()) {
        System.out.println(rs.getString("name"));
        System.out.println(rs.getInt("salary"));
}else {
        System.out.println("No Record Found");
}
```

The **next()** method in the **ResultSet** interface is used to move the cursor to the next row in the result set. It returns a boolean value: **true** if there is a next row, and **false** if there are no more rows. Typically, it is used in a loop to iterate through all the rows in the result set. By default, the cursor is positioned before the first row, and calling next() moves it to the first row. If the result set is empty, next() immediately returns false.

ResultSetMetaData Interface

The **ResultSetMetaData** interface in JDBC provides metadata (information about the structure) of a ResultSet object. It allows developers to retrieve details about the columns in a result set, such as column names, types, sizes, and other properties. This is particularly useful for dynamically handling query results when the column structure is unknown at compile time.

Syntax:

```
ResultSet rs=st.executeQuery("select * from employee");
ResultSetMetaData r=rs.getMetaData();
System.out.println(r.getColumnCount());
System.out.println(r.getColumnName(2));
System.out.println(r.getColumnTypeName(2));
```

Transaction (commit & rollback methods)

In JDBC, **transactions** are a way to group multiple operations (SQL statements) into a single, atomic unit of work. If all operations in the transaction succeed, the changes are committed to the database. If any operation fails, the changes can be rolled back to maintain database consistency. The commit() and rollback() methods of the **Connection** interface are used to manage transactions.

Key Methods

1. **commit()**:
    1. Saves all the changes made by the transaction to the database permanently.
    2. Used after all the statements in a transaction are successfully executed.
2. **rollback()**:
    1. Undoes all the changes made by the transaction, restoring the database to its previous state.
    2. Used when an error occurs, ensuring partial changes are not saved.

Transaction (commit & rollback methods)

Syntax:

```
Statement st=c.createStatement();
c.setAutoCommit(false);
st.executeUpdate("update employee set name='KK' where eid='e102'");
ResultSet rs=st.executeQuery("select * from employee where eid='e102'");
if(rs.next()) {
        System.out.println(rs.getString("eid"));
        System.out.println(rs.getString("name"));
        System.out.println(rs.getInt("salary"));
        System.out.println(rs.getInt("age"));
}
c.rollback();
//c.commit();
rs=st.executeQuery("select * from employee where eid='e102'");
if(rs.next()) {
        System.out.println(rs.getString("eid"));
        System.out.println(rs.getString("name"));
        System.out.println(rs.getInt("salary"));
}
```